Introduction to Artificial Intelligence

丁尧相
浙江大学

Fall & Winter 2022
Week 15

Announcements

- Problem set 4 (last homework) is releasing today. The submission deadline is Jan. 1.
- Lab project 3 is releasing today.
- The final exam is scheduled on Jan. 7. The deadlines of lab projects will be extended to Jan. 7. (recommend to submit before Jan. 1).

Logistic Loss for Logit Linear Model

- The MLE estimator for logistic regression $p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$:

$$\text{MLE}(\theta) = \max_\theta \sum_{i=1}^N \left[\mathbb{I}[y_i=+1]\log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i=-1]\log(\frac{e^{-\mathbf{w}^T\mathbf{x}_i}}{1+e^{-\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \max_\theta \sum_{i=1}^N \left[\mathbb{I}[y_i=+1]\log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i=-1]\log(\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \max_\theta \sum_{i=1}^N \left[\log(\frac{1}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \min_\theta \sum_{i=1}^N \left[\log(1+e^{-y_i\mathbf{w}^T\mathbf{x}_i})\right]$$

logistic loss with logit linear model

$$\max\left(0, 1-y_i(\mathbf{w}^T\mathbf{x}_i+\mathbf{b})\right)$$

hinge loss for linear model

Logistic regression can also be formulated into loss minimization.
Regularization and kernel method can also be used!

Cross-Entropy Loss

- The logistic loss $\log(1+e^{-y\mathbf{w}^T\mathbf{x}})$ in the previous slide is designed for logit linear model in binary classification.

- For general probabilistic model $f(\mathbf{x})$, we can design a similar loss:

$$-\Big[\mathbb{I}[y=+1]log(p)+\mathbb{I}[y=-1]log(1-p)\Big]$$

p is the abbreviation of $p(x)$

This is called the cross-entropy (aka logistic) loss for binary classification.

Multi-Class Cross-Entropy Loss

- For multi-class classification with $y \in \{1, 2, ..., C\}$, extend logistic regression model as

$$p(y = +1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}} \quad \rightarrow \quad p(y_c|\mathbf{x}) = \frac{e^{\mathbf{w}_c^T\mathbf{x}}}{\sum_{i=1}^{C} e^{\mathbf{w}_c^T\mathbf{x}_i}}$$

There is a parameter $\mathbf{w}_c$ for each class.

- The multi-class logistic loss for the above model is

$$\log(1 + e^{-y\mathbf{w}^T\mathbf{x}}) \quad \rightarrow \quad -\sum_{c=1}^{C} \mathbb{I}[y = y_c] \log p(y_c|\mathbf{x})$$

- In general, $p(y_c|\mathbf{x})$ can represented by other functions, then the multi-class cross-entropy loss is also

$$-\sum_{c=1}^{C} \mathbb{I}[y = y_c] \log p(y_c|\mathbf{x})$$

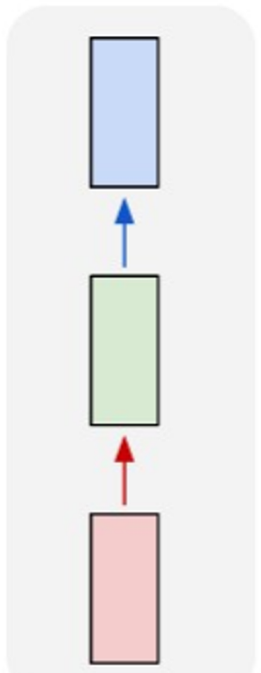
Usually the most common choice for classification with NN.

Machine Learning: VI

- Reinforcement learning
  - Review of Basics
  - Function approximation
  - Policy gradient & actor-critic methods
- Deep reinforcement learning
- Integrating learning and planning
- Take-home messages

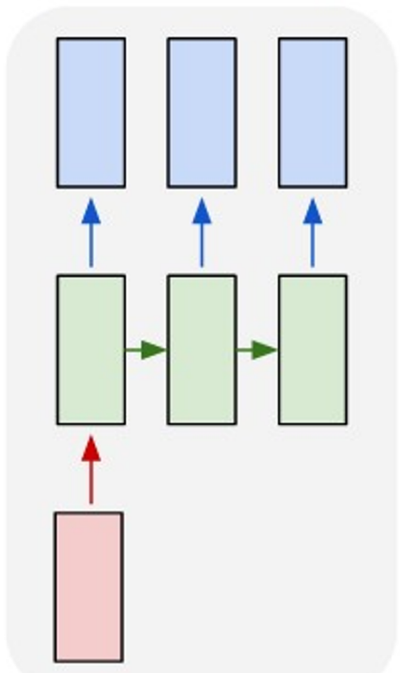
Sequential Learning Problems

one-to-one



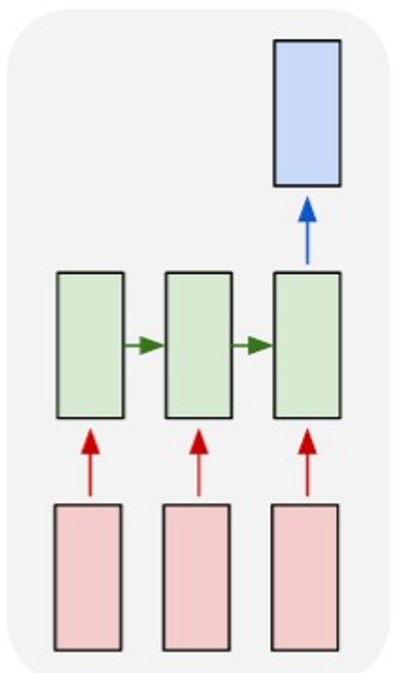
e.g.
image
classification

one-to-many



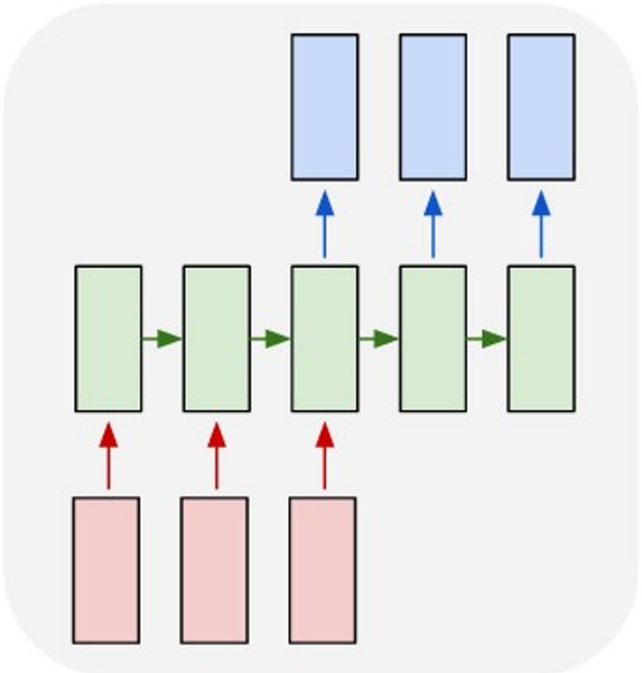
e.g.
image captioning:
image ->sequence of words

many-to-one

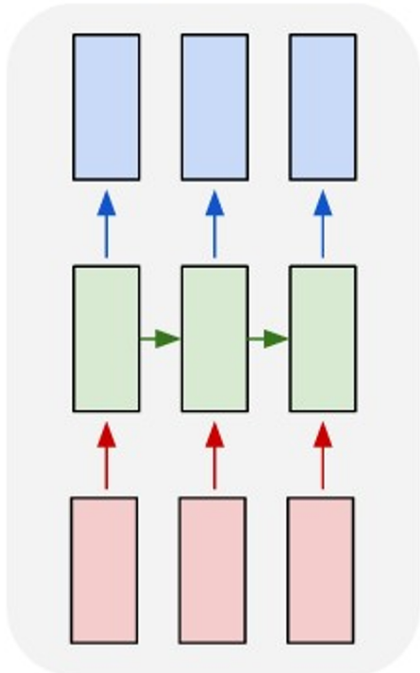


e.g.
video classification:
sequence of images -> label

many-to-many

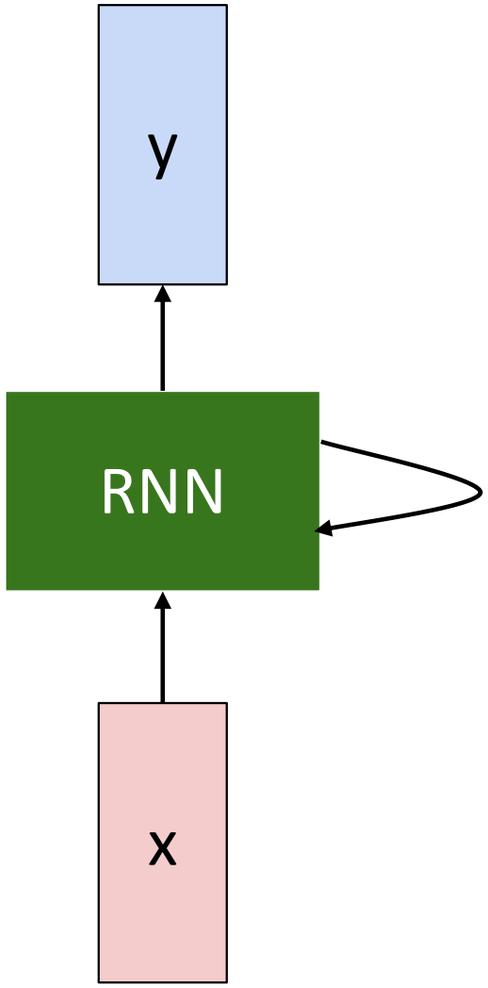


e.g.
machine translation:
sequence of words
->
sequence of words



e.g.
per-frame
video classification
sequence of images
->
sequence of images

Recurrent Neural Networks



We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step

Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Unsupervised Learning

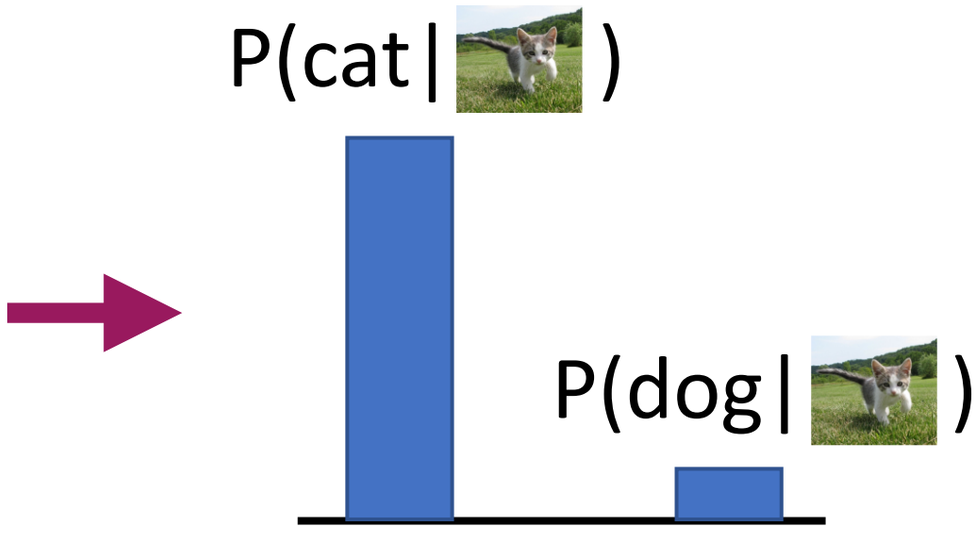
Data: x

Just data, no labels!

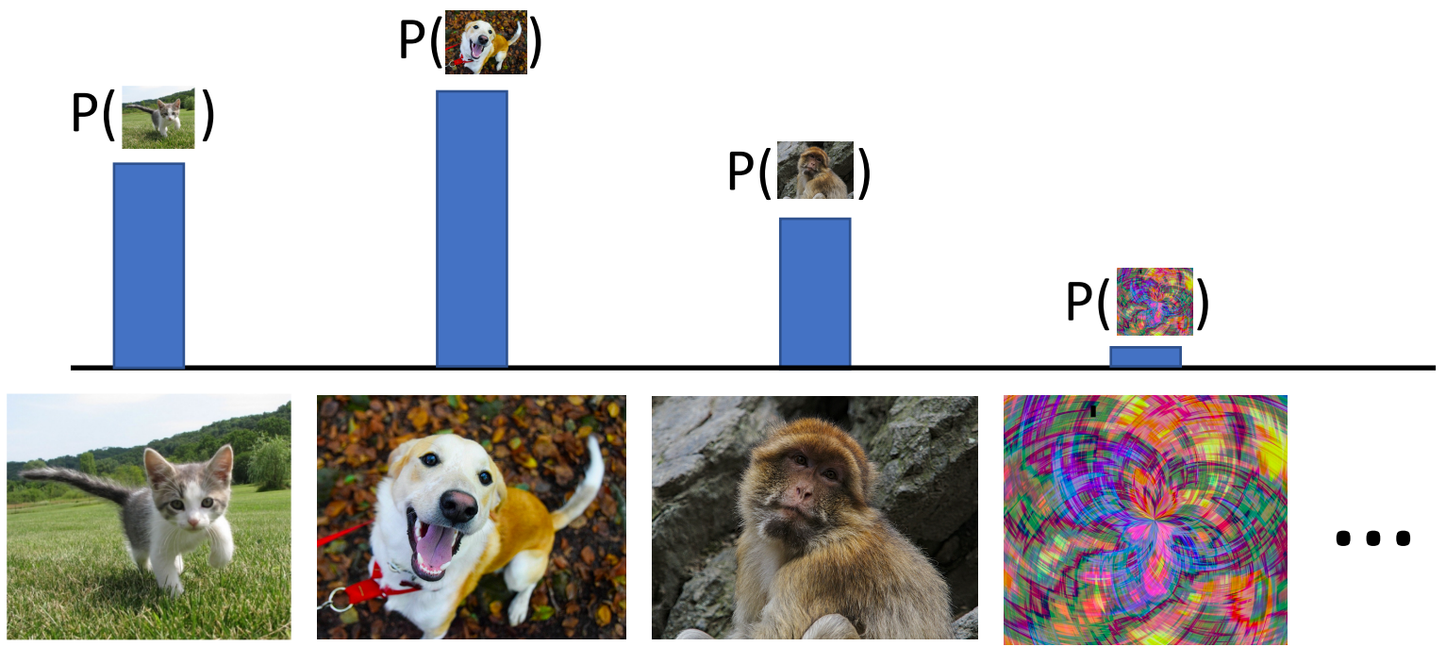
Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Generative Models



Discriminative Model: Learn a probability distribution p(y|x)

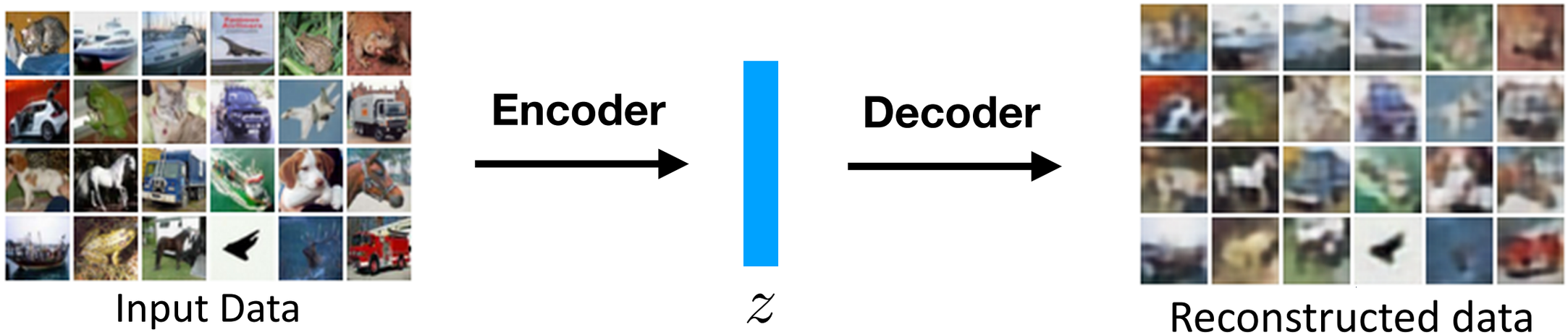


Generative model: All possible images compete with each other for probability mass

Generative Model: Learn a probability distribution p(x)

Autoencoder

- An autoencoder consists of both an encoder and a decoder:
  - Encoder: transform input x into latent representation z
  - Decoder: generate recovered input $\hat{x}$ from z

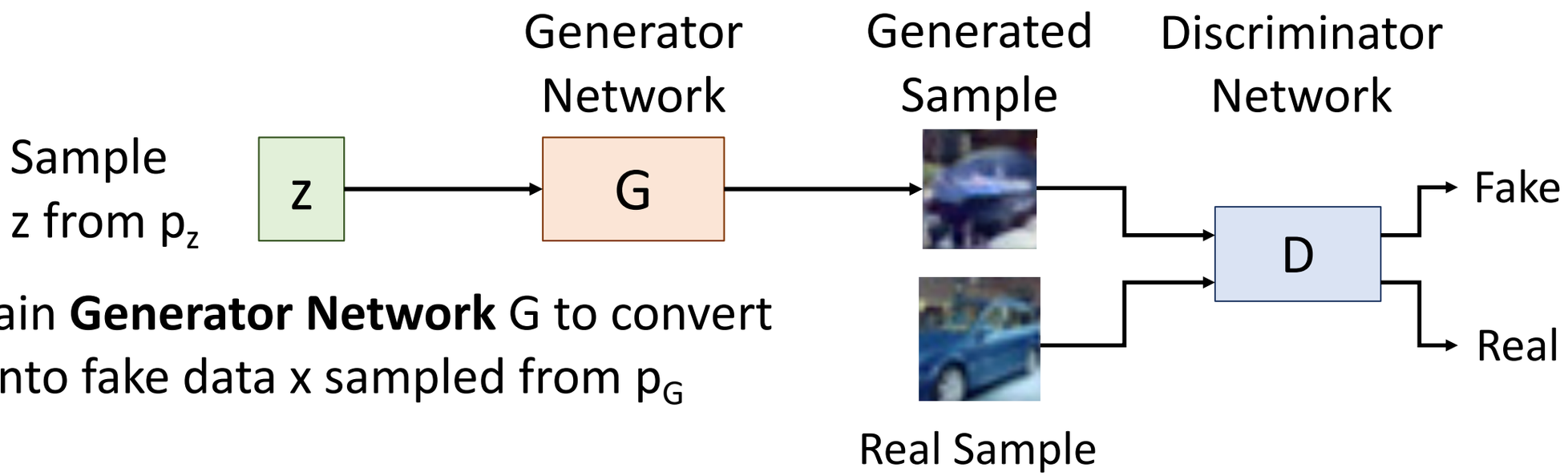


The targets are two-fold:
learn good encoder to compress the information.
learn good decoder to recover the information.

Generative Adversarial Networks

- Target: obtain a model for $p(x)$, then we can sample data from it.

Idea: Introduce a latent variable z with simple prior p(z).
Sample $z \sim p(z)$ and pass to a **Generator Network** x = G(z)
Then x is a sample from the **Generator distribution** p_G. Want $p_G = p_{data}$!



Train **Generator Network** G to convert z into fake data x sampled from p_G

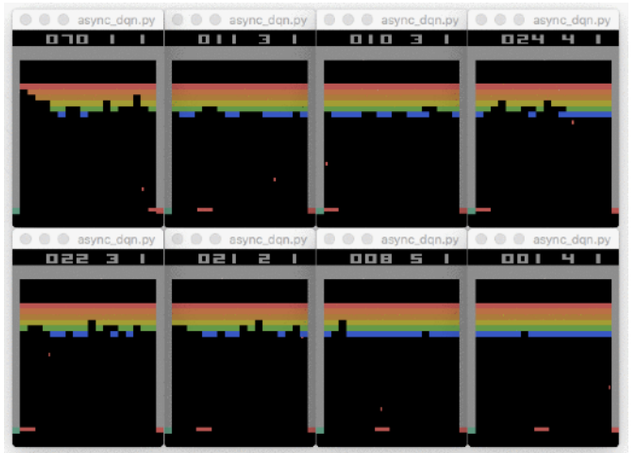
The key idea is to train a discriminator to classify fake and real data.
A good generator should fool the discriminator to make its accuracy low:

$$p_G = p_{data}$$

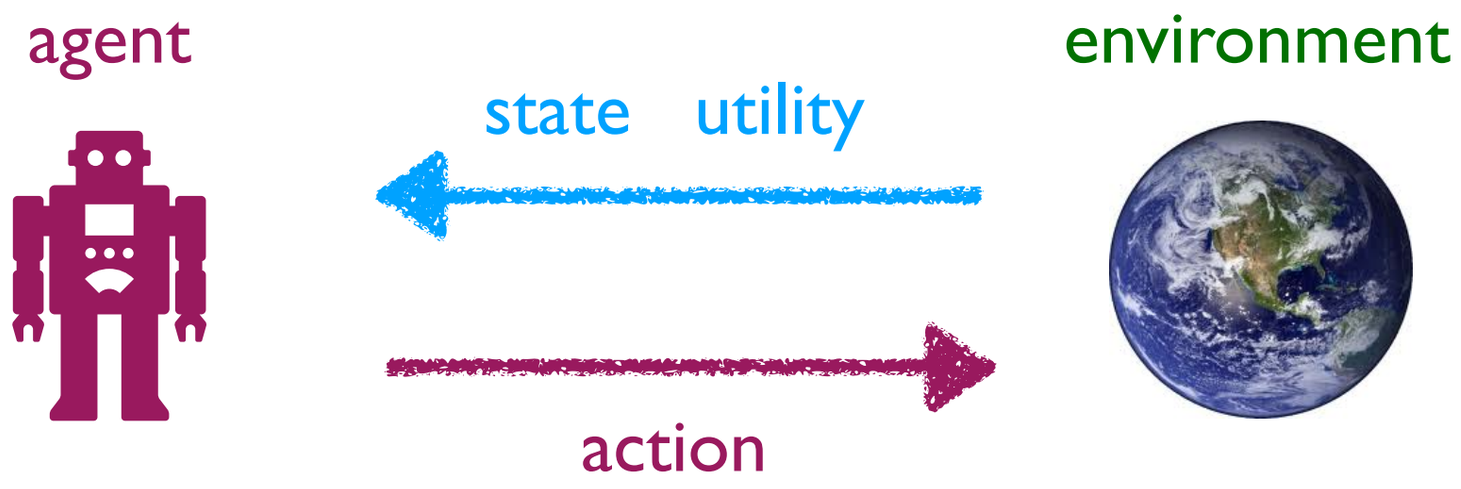
Machine Learning:VI

- Reinforcement learning
  - Review of Basics
  - Function approximation
  - Policy gradient & actor-critic methods
  - Deep reinforcement learning
  - Integrating learning and planning
  - Take-home messages

Decision Making

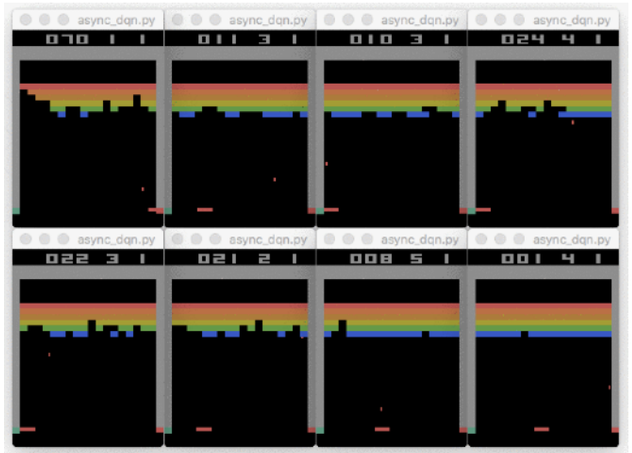


- Conduct action in any state of an environment.

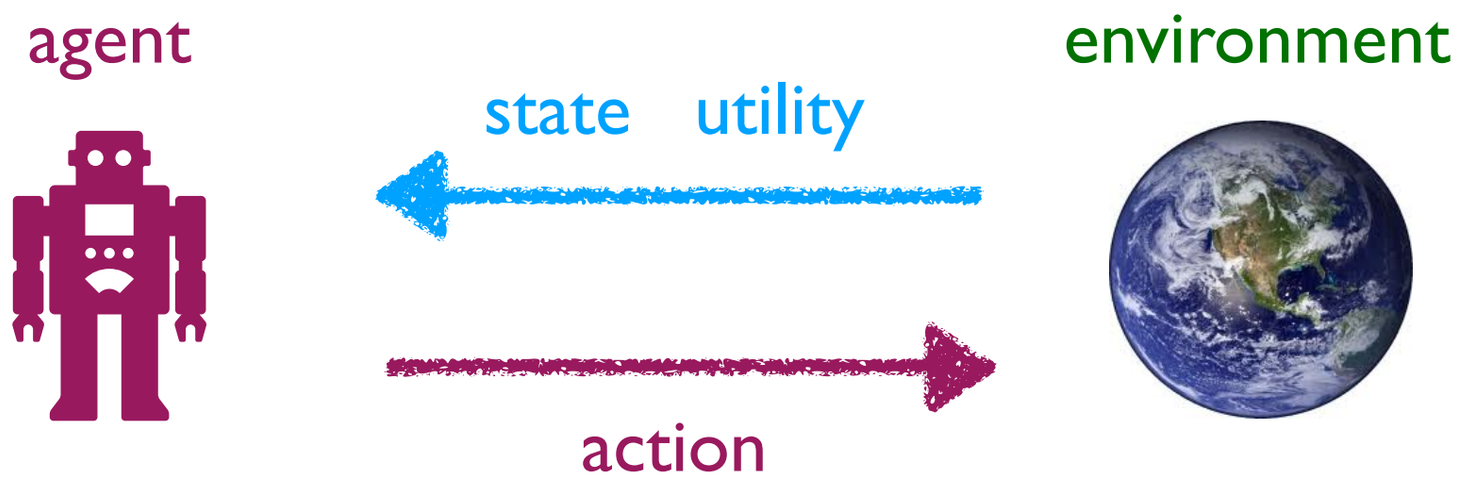


In most problems, the agent needs to do a sequence of actions w.r.t. a sequence of states.

Decision Making

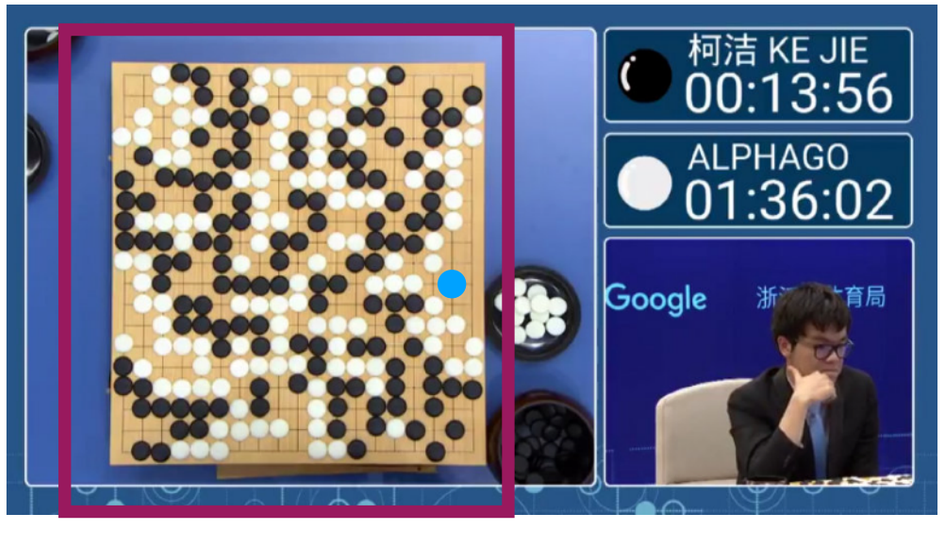


- Conduct action in any state of an environment.



In most problems, the agent needs to do a sequence of actions w.r.t. a sequence of states.

The Decision Problem

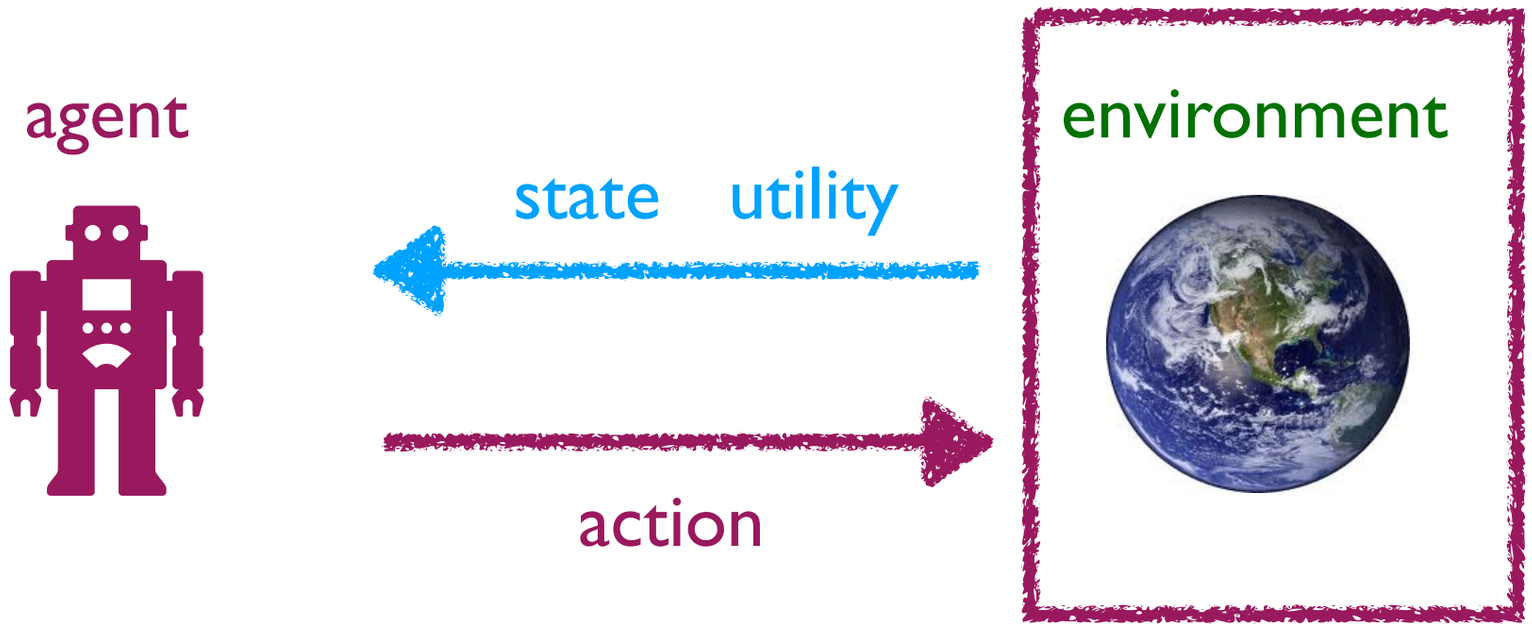


- The agent faces with a series of "states".
- Need to choose the corresponding "actions".
- Each action has a utility/cost.

We give utility a name: reward.

- Target: maximize the total reward in a decision sequence by always choosing the right action.

Model of the Environment



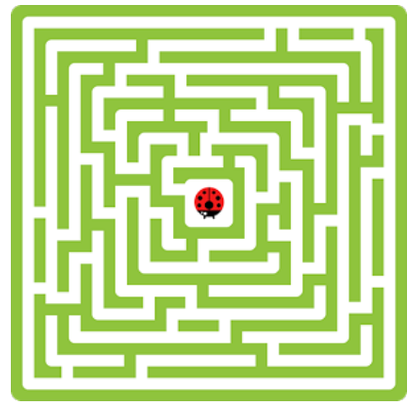
To make decisions in the environment, the agent usually needs a model of the environment to know how the things go on.
Where does this model come from?
Given by the problem (external) or built by the agent? (internal)

Reinforcement Learning

- Decision making is to find the optimal policy:
    - Decide best actions on all states.
    - No labeled <state, action> data, only receive reward.
        - The target is to maximize the long term total reward.
- Search-based decision making:
    - When the model is known, and the search cost is reasonable.
- Reinforcement learning:
    - Decision making in unknown model or search cost is too high.

Markov Decision Process

- How to model a maze problem?
- State: the current position.
- Action: left, right, up, down.



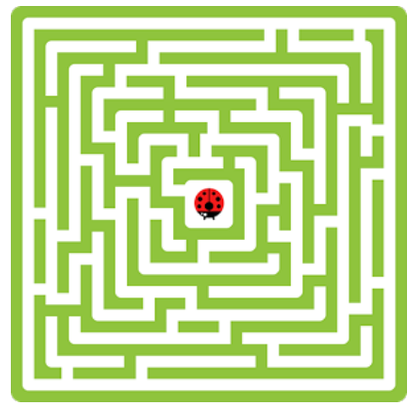
- Transition: where is the next position when take an action?
- Reward: how good is it instantly when take an action?
- Discount factor: How much the current action influences future?

Markov decision process (MDP) is the decision making model in RL with specific assumptions.

Mathematical Formulation

- A Markov Decision Process (MDP) is a five-tuple

$$< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$$



- $\mathcal{S}$ — The space of possible states (cont. or discrete)
- $\mathcal{A}$ — The space of possible actions (cont. or discrete)
- $\mathcal{P} : p(s_{t+1}|s_t, a_t)$ — The transition function (distribution)
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ — The reward function
- γ — The discount factor of rewards

The transition and reward functions can be stochastic!

The Markov Property

"The future is independent of the past given the present"

- The Markov property:

$$p(s_{t+1}|s_t, a_t) = p(s_{t+1}|s_1, s_2, ...s_t, a_t)$$

- The next state is only decided by the current state and action.
- The current state is a sufficient statistic.
- Non-Markovian decision problem:

小张出生于中国，2016年来到美国留学。小张的母语是(？)

The Learning Agent

- The agent takes a series of actions, experiences a series of states, and receives a series of rewards:

$$\{s_1, a_1, r_1\}, \{s_2, a_2, r_2\}, \{s_3, a_3, r_3\}...$$

- policy: function $p(a|s)$ used to select actions on any states.

- The target is to find the optimal policy to maximize the discounted total reward along the timeline:

$$r_1 + \gamma r_2 + \gamma^2 r_3 + ... = \sum_{t=1}^{\infty} \gamma^{t-1} r_t$$

- The discount factor measures how much the long term effect of the current action is concerned.

Value Functions: State Value Function V

- The state value function of a given policy is the expected total reward start from a given state, then follow the policy:

$$V_\pi(s) = \mathbb{E}_{\pi, p(s|s,a)}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right]$$

- The optimal policy have the optimal value function:

$$V_{\pi^*}(s) = \max_\pi V_\pi(s)$$

Value Functions: Action-State Value Function Q

- The action-state value function of a given policy is the expected total reward start from a given state, execute a given action, then follow the policy:

$$Q_\pi(s,a) = \mathbb{E}_{\pi, p(s|s,a)}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a\right]$$

- The optimal deterministic policy chooses the optimal action:

$$\pi^*(s) = \arg\max_a Q_{\pi^*}(s,a)$$

If the optimal action-state value function is known, so is the optimal policy!

Bellman Equation

- For the state value function,

$$V_\pi(s) = \mathbb{E}_{\pi(s),p(s|s,a)}\Big[\sum_{t=0}^{\infty}\gamma^t r_t|s_0 = s\Big]$$

$$= \mathbb{E}_{\pi(s_0),p(s_1|s_0,a_0)}\Big[r_0 + \gamma\mathbb{E}_{\pi(s),p(s|s,a)}\big[\sum_{t=1}^{\infty}\gamma^{t-1} r_t|s_1\big]|s_0 = s\Big]$$

$$= \mathbb{E}_{\pi(s_0),p(s_1|s_0,a_0)}\Big[r_0 + \gamma V_\pi(s_1)|s_0 = s\Big]$$

Recursive Definition

- For discrete state and action, and deterministic policy,

$$V_\pi(s) = \sum_{s'} p\big(s'|s,\pi(s)\big)\Big[r\big(s,\pi(s),s'\big) + \gamma V_\pi(s')\Big]$$

Bellman Equation (Cont.)

- For the action-state value function,

$$Q_\pi(s,a) = \mathbb{E}_{\pi(s),p(s|s,a)}\Big[\sum_{t=0}^{\infty}\gamma^t r_t|s_0=s,a_0=a\Big]$$

$$= \mathbb{E}_{\pi(s_0),p(s_1|s_0,a_0)}\Big[r_0+\gamma\mathbb{E}_{\pi(s),p(s|s,a)}\big[\sum_{t=1}^{\infty}\gamma^{t-1}r_t|s_1,a_1\big]|s_0=s,a_0=a\Big]$$

$$= \mathbb{E}_{\pi(s_0),p(s_1|s_0,a_0)}\Big[r_0+\gamma Q_\pi(s_1,a_1)|s_0=s,a_0=a\Big]$$

Recursive Definition

- For discrete state and action, and deterministic policy,

$$Q_\pi(s,a) = \sum_{s'} p\big(s'|s,a\big)\Big[r(s,a,s')+\gamma Q_\pi\big(s',\pi(s')\big)\Big]$$

Bellman Equation (Cont.)

- For optimal deterministic policy π^*,

$$V_{\pi^*}(s) = \max_a \Big[\sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V_{\pi^*}(s')] \Big]$$

$$Q_{\pi^*}(s,a) = \sum_{s'} p(s'|s,a) \Big[r(s,a,s') + \gamma \max_{a'} Q_{\pi^*}(s',a') \Big]$$

- Then the optimal deterministic policy is

$$\pi^*(s) = \arg\max_a Q_{\pi^*}(s,a)$$

- Due the recursive structure, the optimal value functions can be solved by dynamical programming. This assumes that the full information of the MDP is known!

Value Iteration

- Initialize value function V_0
- For i=1,2,3… until convergence
    - Update V_i for each state

Why iterative update?
Loop exists in the MDP!

$$V_i(s) = \max_a \left[\sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V_{i-1}(s')] \right]$$

- Theoretical convergence guarantee to V^* and π^*

Policy Iteration

- Initialize value function V_0 and policy π_0
- For i=1,2,3… until convergence
    - Policy evaluation step: update V_i for each state

$$V_i(s) = r + \gamma V_{i-1}^{\pi_{i-1}}(s')$$

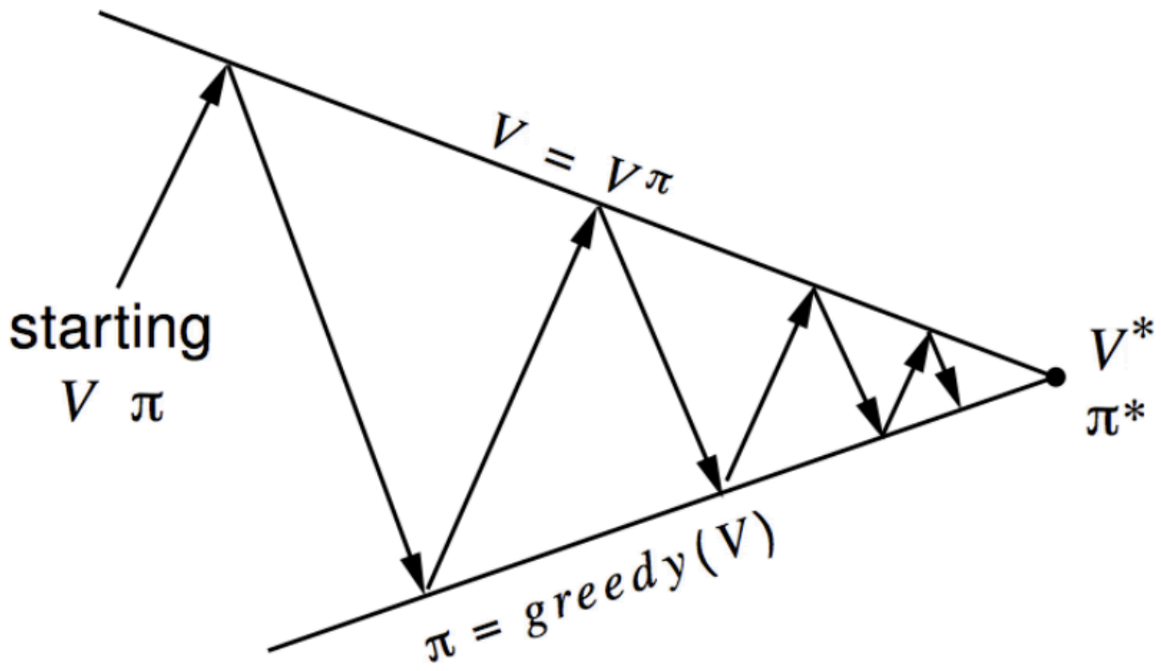
Calculated based on π_{i-1}
Actually an inner loop to do iterative update until convergence

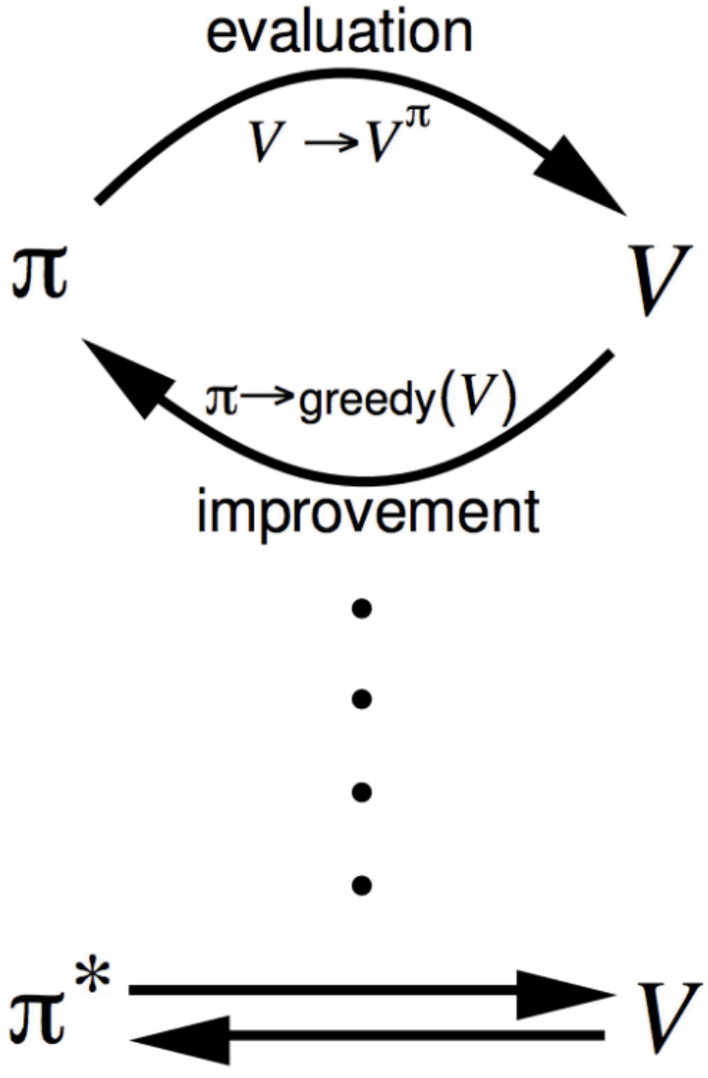
    - Policy improvement step: update π_i for each s-a pair.

$$\pi(s) \leftarrow arg\max_a \sum_{s',r'} p(s', r|s, a)[r + \gamma V(s')]$$

- Theoretical convergence guarantee to V^* and π^*

Policy Iteration (Cont.)





Policy evaluation Estimate v_π
Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$
Greedy policy improvement

Slide courtesy: David Silver

Reinforcement Learning

- When full information of the MDP is known, the value function can be solved by planning.

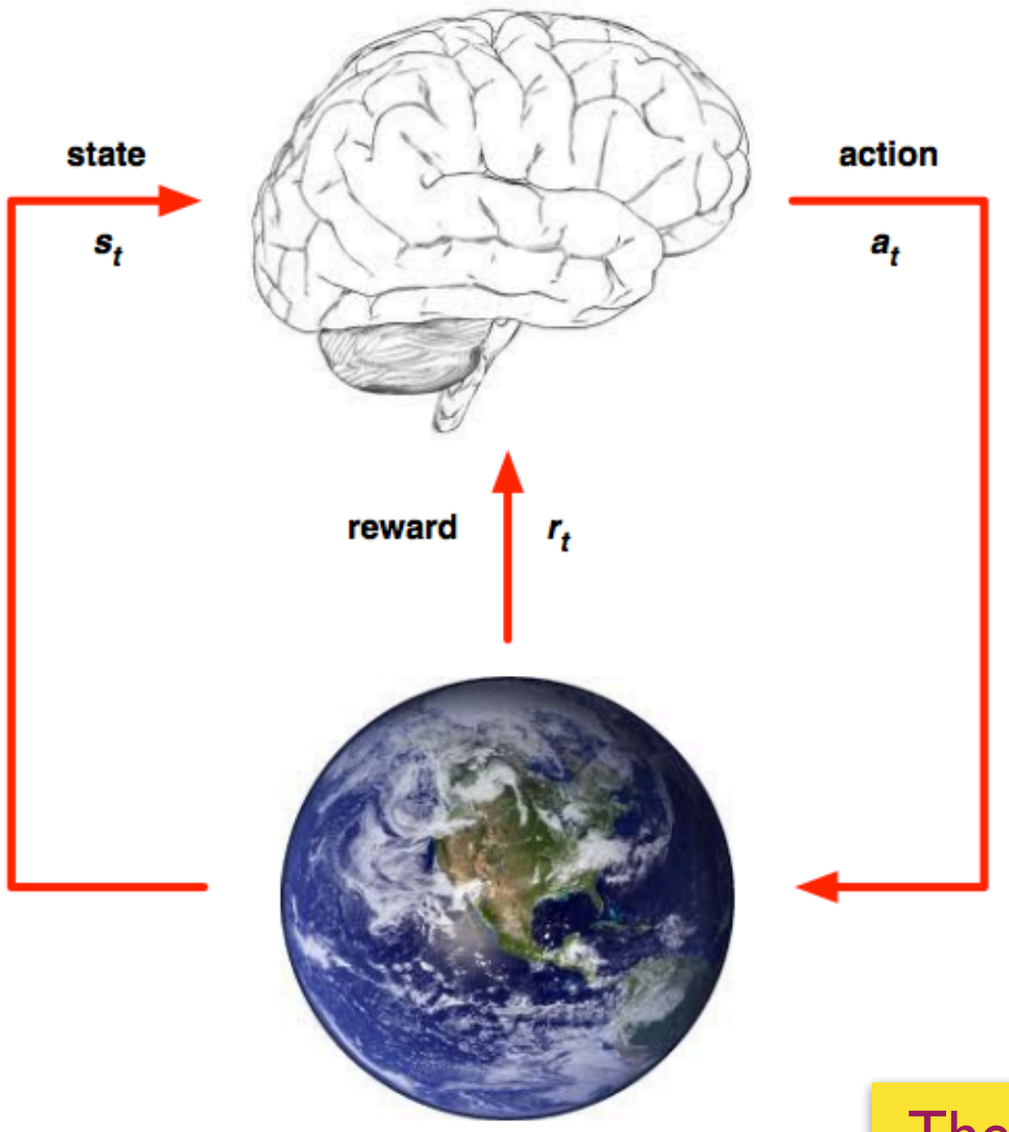
- But how to solve when not fully known? $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$

- In RL, usually the state transition $\mathcal{P}$ and reward function $\mathcal{R}$ are not known.

- The agent has to learn by trial and error, facing with the exploration and exploitation problem.

Collect data by the agent itself during learning.

Agent and Environment



- At each step t the agent:
  - Receives state s_t
  - Receives scalar reward r_t
  - Executes action a_t
- The environment:
  - Receives action a_t
  - Emits state s_t
  - Emits scalar reward r_t

The agent can only interact with true environment.
Can not use explicit model for searching or planning.
But can learn internal model!

- The target is still to learn the optimal value function.

Basic Idea

- Value function based RL aims at estimating the optimal value function.

- Value function update: update using new estimation

$$Q_{i+1}(s,a) = (1-\alpha)Q_i(s,a) + \alpha\hat{Q}_i(s,a)$$

  - Monte-Carlo RL — Estimate by sampled trajectories

  - Temporal difference Learning — SARSA and Q-learning.

- Policy Improvement:

  - Based on new value function, with ϵ - greedy.

Monte-Carlo RL

- Given policy π_i, we can sample trajectories:

$$\{s_1, a_1, r_1\}, \{s_2, a_2, r_2\}, \{s_3, a_3, r_3\}...$$

- Then we can get empirical estimate:

$$\hat{Q}_i(s_1, a_1) = r_1 + \gamma r_2 + \gamma^2 r_3 + ...$$

Can we still update the policy in greedy?

- Update value function:

$$Q_{i+1}(s, a) = (1 - \alpha)Q_i(s, a) + \alpha\hat{Q}_i(s, a)$$

No!

- Follow the spirit of policy iteration, update $\pi_i \rightarrow \pi_{i+1}$

Exploration vs. Exploitation

"Behind one door is tenure - behind the other is flipping burgers at McDonald's."

- There are two doors in front of you.
- You open the left door and get reward 0
  $V(left) = 0$
- You open the right door and get reward $+1$
  $V(right) = +1$
- You open the right door and get reward $+3$
  $V(right) = +2$
- You open the right door and get reward $+2$
  $V(right) = +2$

$\vdots$

- Are you sure you've chosen the best door?

Exploration vs. Exploitation (Cont.)

- In policy iteration, the policy improvement step is greedy:

$$\pi_i(s) = \arg\max_a Q_i(s, a)$$

- But for RL, since the environment is not fully known, greedy update may perform arbitrarily bad — need to allow some exploration.

- Common choice: use ϵ- greedy policy:

with prob. $1 - \epsilon$, execute as greedy

with prob. ϵ, execute randomly

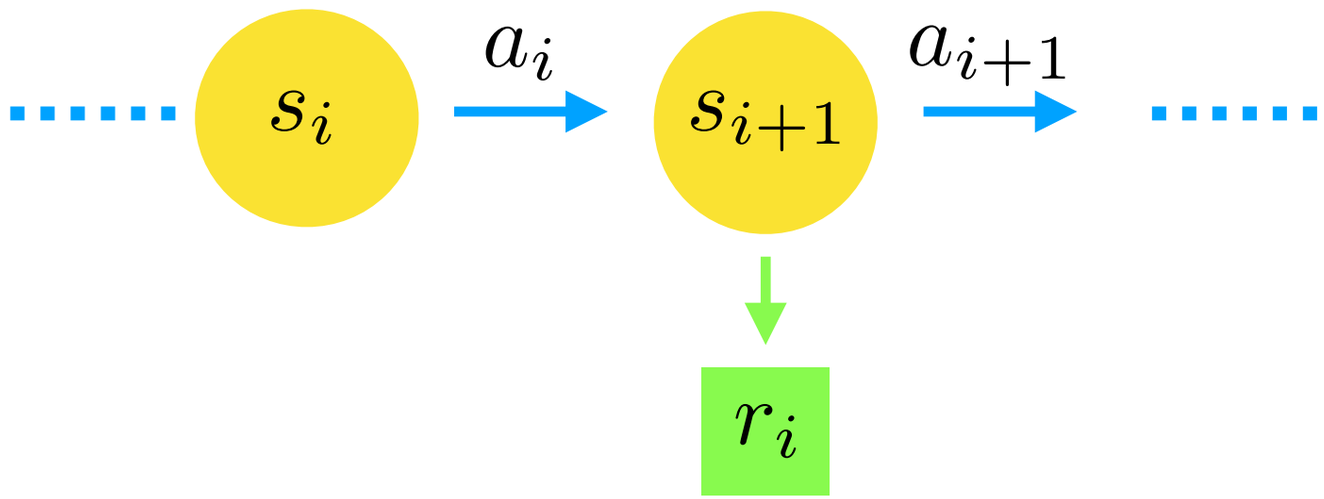
- Theoretical guarantee: If the exploration vanishes, we can ensure convergence.

Temporal Difference vs. Monte-Carlo

- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Incomplete sequences
- Natural idea: use TD instead of MC in our control loop
  - Apply TD to $Q(S, A)$
  - Use ϵ-greedy policy improvement
  - Update every time-step

SARSA

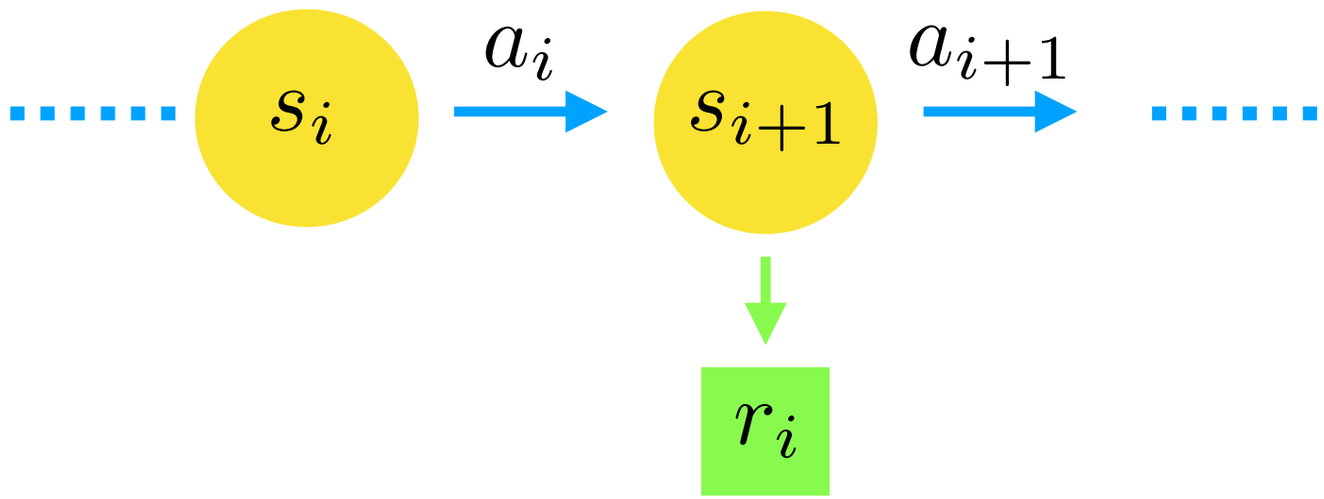
- "State-Action-Reward-State-Action" — SARSA



Run the current best ϵ-greedy policy

SARSA

- "State-Action-Reward-State-Action" — SARSA



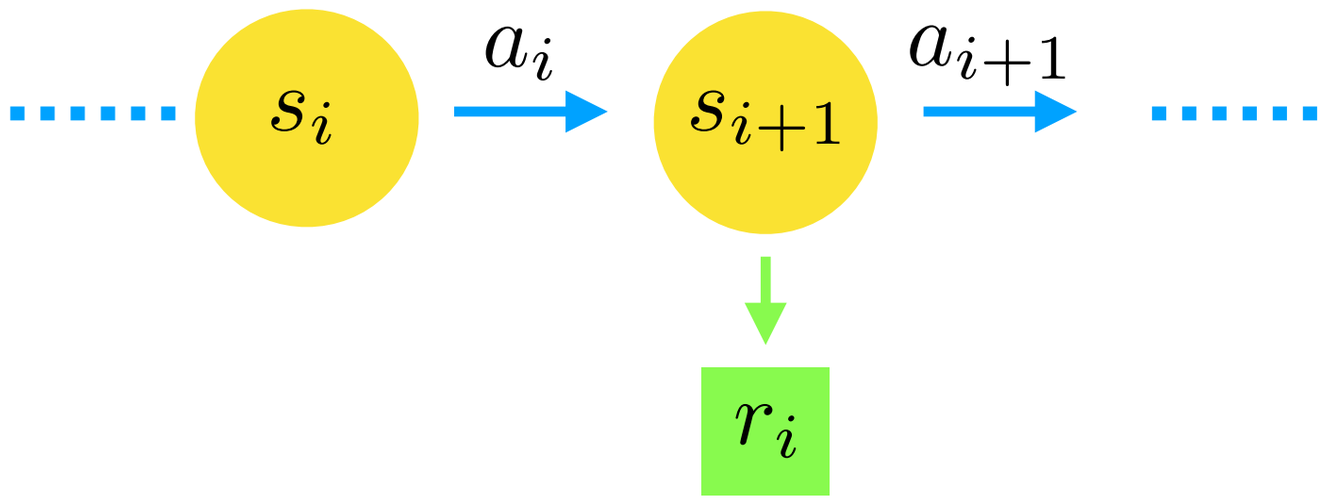
Run the current best ϵ-greedy policy

- Once collect s-a-r-s-a sample, do value function update:

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \Big[r_i + \gamma Q(s_{i+1}, a_{i+1}) - Q(s_i, a_i) \Big]$$

SARSA

- "State-Action-Reward-State-Action" — SARSA



Run the current best ϵ-greedy policy

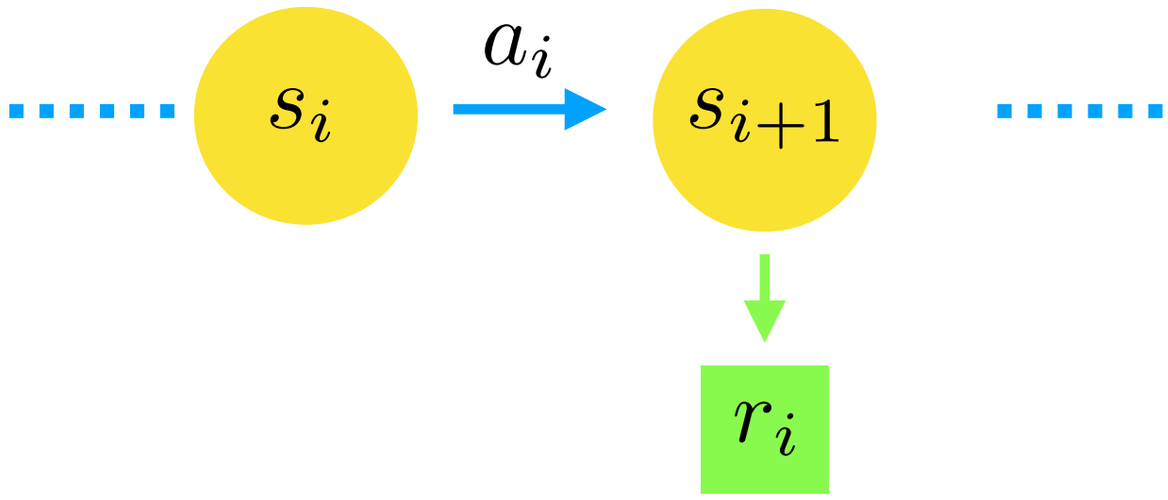
- Once collect s-a-r-s-a sample, do value function update:

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \Big[r_i + \gamma Q(s_{i+1}, a_{i+1}) - Q(s_i, a_i) \Big]$$

TD error

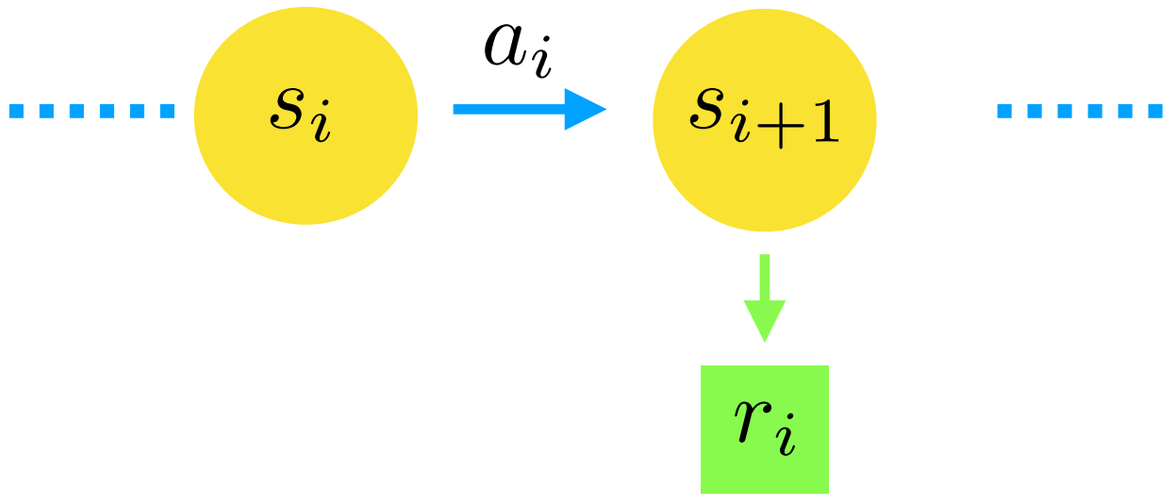
Always use the policy on-the-run, called "on-policy"

Q-Learning



Run the current
best ϵ-greedy policy

Q-Learning



Run the current
best ϵ-greedy policy

- Once collect s-a-r-s sample, do value function update:

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha\Big[r_i + \gamma Q(s_{i+1}, \hat{\pi}^*(s_{i+1})) - Q(s_i, a_i)\Big]$$

The current best policy

The policy on the run can be different from
the current best policy in the update, called "off-policy"

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s,a)$
- While following behaviour policy $\mu(a|s)$

$$\{S_1, A_1, R_2, ..., S_T\} \sim \mu$$

- Why is this important?
- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, ..., \pi_{t-1}$
- Learn about *optimal* policy while following *exploratory* policy
- Learn about *multiple* policies while following *one* policy

Large-Scale RL

- Large decision-making problems:
  - Backgammon: 10^{20} states
  - Go: 10^{170} states
  - Robot control: continuous state space

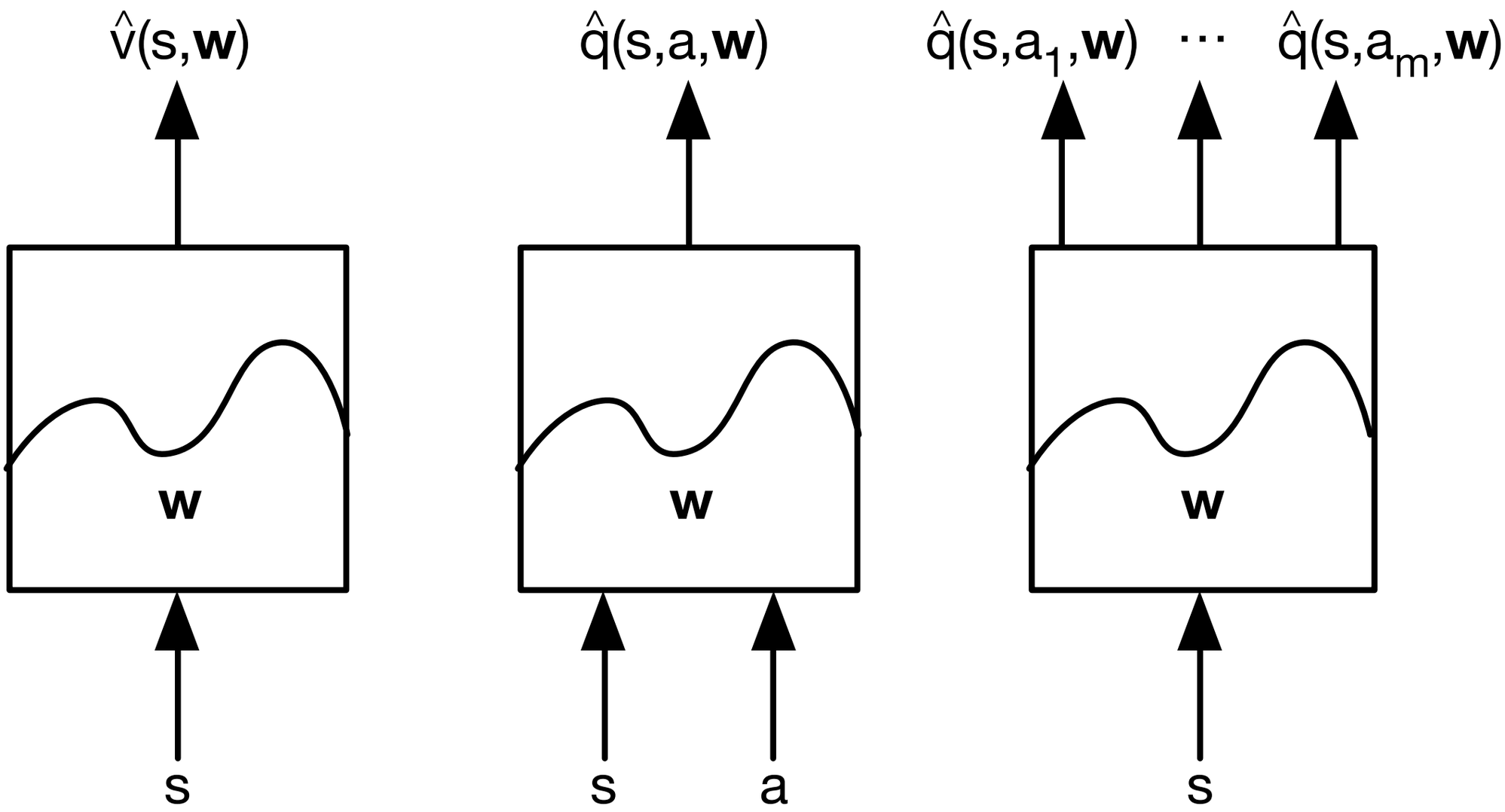
Classic value function methods rely on tabular representation of value functions.
Obviously needing more compact representations.

Machine Learning:VI

- Reinforcement learning
  - Review of Basics
  - Function approximation
  - Policy gradient & actor-critic methods
  - Deep reinforcement learning
  - Integrating learning and planning
  - Take-home messages

Types of Value Function Approximation

output: value function scores



input: state or/and actions

Feature Vectors

- Represent state by a *feature vector*

$$\mathbf{x}(S) = \begin{pmatrix} \mathbf{x}_1(S) \\ \vdots \\ \mathbf{x}_n(S) \end{pmatrix}$$

- For example:
  - Distance of robot from landmarks
  - Trends in the stock market
  - Piece and pawn configurations in chess

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S) \mathbf{w}_j$$

- Objective function is quadratic in parameters $\mathbf{w}$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w})^2 \right]$$

- Stochastic gradient descent converges on *global* optimum
- Update rule is particularly simple

$$\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \mathbf{x}(S)$$

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \mathbf{x}(S)$$

Update $=$ *step-size* $\times$ *prediction error* $\times$ *feature value*

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S)\mathbf{w}_j$$

- Objective function is quadratic in parameters $\mathbf{w}$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w})^2 \right]$$

unknown true value.
need to estimate during learning!

- Stochastic gradient descent converges on *global* optimum
- Update rule is particularly simple

$$\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \mathbf{x}(S)$$

$$\Delta \mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S, \mathbf{w}))\mathbf{x}(S)$$

Update = *step-size* $\times$ *prediction error* $\times$ *feature value*

Linear Value Function Approximation

- Represent value function by a linear combination of features

$$\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S)\mathbf{w}_j$$

- Objective function is quadratic in parameters $\mathbf{w}$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w})^2 \right]$$

unknown true value.
need to estimate during learning!

- Stochastic gradient descent converges on *global* optimum
- Update rule is particularly simple

$$\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \mathbf{x}(S)$$

$$\Delta \mathbf{w} = \alpha(v_\pi(S) - \hat{v}(S, \mathbf{w}))\mathbf{x}(S)$$

Update = *step-size* $\times$ *prediction error* $\times$ *feature value*

beyond simple linear regression

Function Approximators

There are many function approximators, e.g.

- Linear combinations of features
- Neural network
- Decision tree
- Nearest neighbour
- Fourier / wavelet bases
- ...

Function Approximators

There are many function approximators, e.g.

- Linear combinations of features
- Neural network

more commonly used

- Decision tree
- Nearest neighbour
- Fourier / wavelet bases
- ...

Function Approximators

There are many function approximators, e.g.

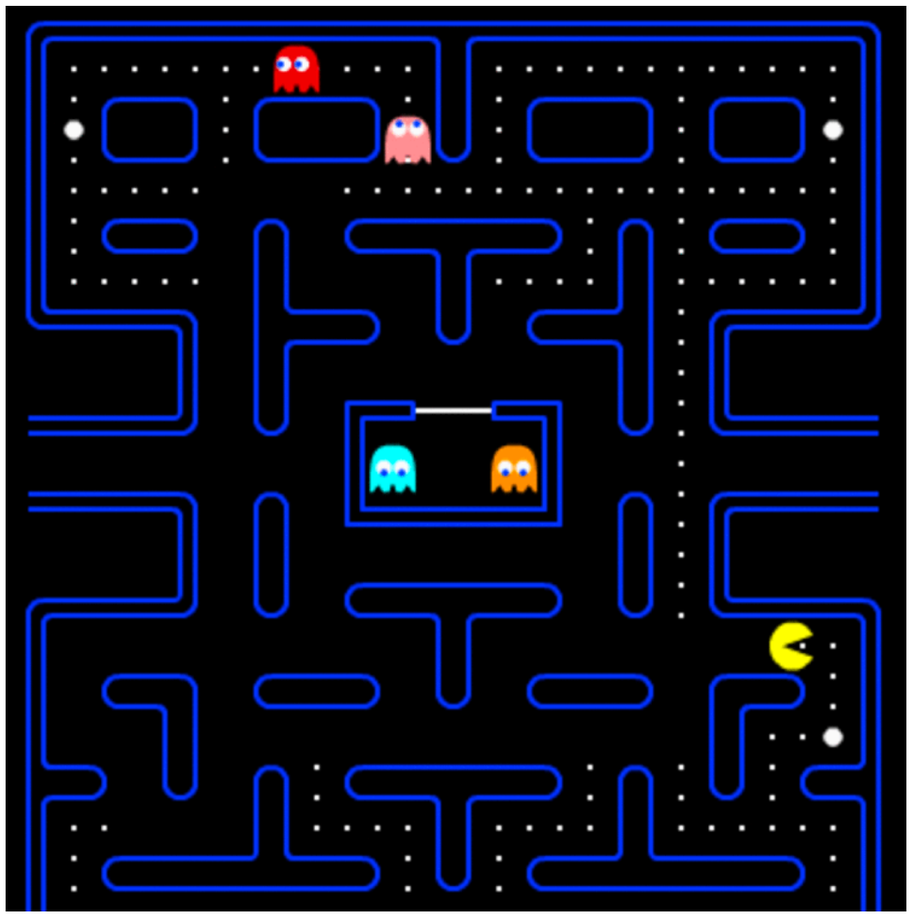
- Linear combinations of features
- Neural network

more commonly used

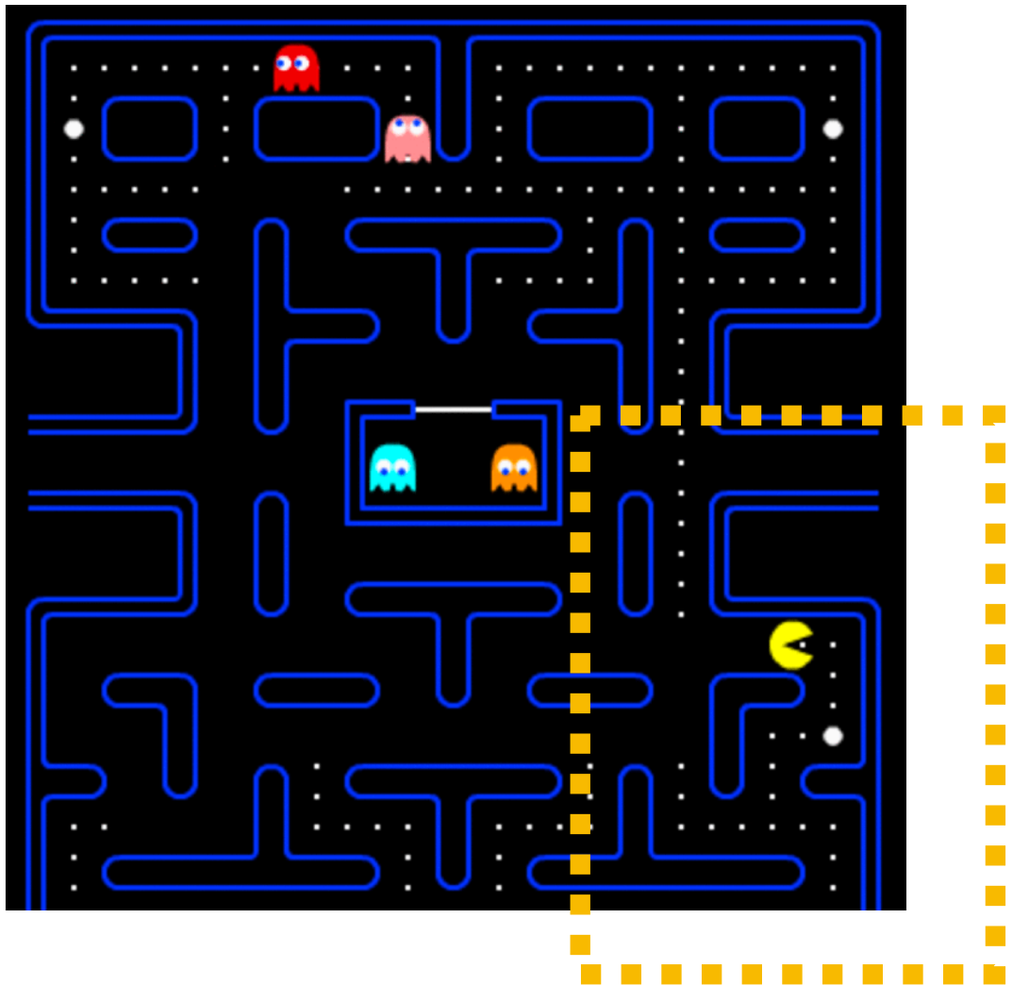
- Decision tree
- Nearest neighbour
- Fourier / wavelet bases
- ...

Different from traditional supervised learning,
we need learning algorithms that can handle data collected by the learner online:
biased and unstable.

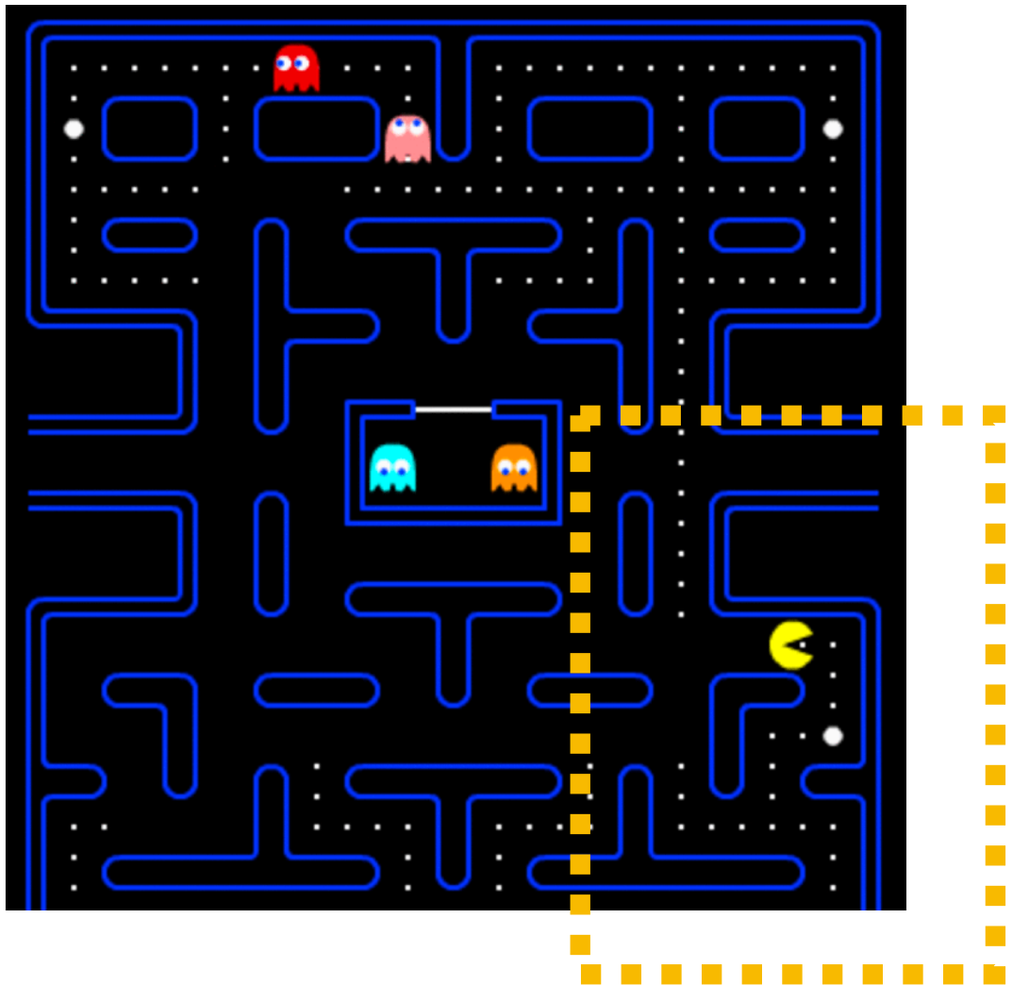
Bias and Instability



Bias and Instability



Bias and Instability



The issue of bias and instability for data collection lies at the heart of RL.
This is also why we need exploration.

Approximate Targets via Bellman Equation

- Have assumed true value function $v_\pi(s)$ given by supervisor
- But in RL there is no supervisor, only rewards

- In practice, we substitute a *target* for $v_\pi(s)$
  - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

  - For TD(0), the target is the TD target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

  - For TD(λ), the target is the λ-return G_t^λ

$$\Delta \mathbf{w} = \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

Approximate Targets via Bellman Equation

- Have assumed true value function $v_\pi(s)$ given by supervisor
- But in RL there is no supervisor, only rewards

- In practice, we substitute a *target* for $v_\pi(s)$
  - For MC, the target is the return G_t

Monte-Carlo estimation

$$\Delta \mathbf{w} = \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

  - For TD(0), the target is the TD target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

  - For TD(λ), the target is the λ-return G_t^λ

$$\Delta \mathbf{w} = \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

Approximate Targets via Bellman Equation

- Have assumed true value function $v_\pi(s)$ given by supervisor
- But in RL there is no supervisor, only rewards

- In practice, we substitute a *target* for $v_\pi(s)$
  - For MC, the target is the return G_t

Monte-Carlo estimation

$$\Delta \mathbf{w} = \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

  - For TD(0), the target is the TD target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

  - For TD(λ), the target is the λ-return G_t^λ

$$\Delta \mathbf{w} = \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

Temporal-difference estimation

Action Value Function Approximation

- Approximate the action-value function

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$$

- Minimise mean-squared error between approximate action-value fn $\hat{q}(S, A, \mathbf{w})$ and true action-value fn $q_\pi(S, A)$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2 \right]$$

- Use stochastic gradient descent to find a local minimum

$$-\frac{1}{2} \nabla_{\mathbf{w}} J(\mathbf{w}) = (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w})$$

Linear Action-Value Function Approximation

- Represent state *and* action by a *feature vector*

$$\mathbf{x}(S, A) = \begin{pmatrix} \mathbf{x}_1(S, A) \\ \vdots \\ \mathbf{x}_n(S, A) \end{pmatrix}$$

- Represent action-value fn by linear combination of features

$$\hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S, A)\mathbf{w}_j$$

- Stochastic gradient descent update

$$\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)$$

$$\Delta\mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\mathbf{x}(S, A)$$

Linear Action-Value Function Approximation

- Like prediction, we must substitute a *target* for $q_\pi(S, A)$
  - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

  - For TD(0), the target is the TD target $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

- For forward-view TD(λ), target is the action-value λ-return

$$\Delta \mathbf{w} = \alpha(q_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

- For backward-view TD(λ), equivalent update is

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}) \\
E_t &= \gamma\lambda E_{t-1} + \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}) \\
\Delta \mathbf{w} &= \alpha \delta_t E_t
\end{aligned}$$

Linear Action-Value Function Approximation

- Like prediction, we must substitute a *target* for $q_\pi(S, A)$
  - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

  - For TD(0), the target is the TD target $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

SARSA here. Can also do Q-learning (more later)

- For forward-view TD(λ), target is the action-value λ-return

$$\Delta \mathbf{w} = \alpha(q_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

- For backward-view TD(λ), equivalent update is

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}) \\
E_t &= \gamma\lambda E_{t-1} + \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w}) \\
\Delta \mathbf{w} &= \alpha \delta_t E_t
\end{aligned}$$

Machine Learning: VI

- Reinforcement learning
  - Review of Basics
  - Function approximation
  - Policy gradient & actor-critic methods
  - Deep reinforcement learning
  - Integrating learning and planning
  - Take-home messages

Direct Policy Learning

- For value function based RL, policy is not directly optimized.
    - Not capable to learn in continuous state and action space.
    - Not capable to learn stochastic policy.
    - May not learn fast.

- Can learn stochastic policy directly:
    - Parametrize policy $\pi_\theta(a|s)$ with parameter θ
        - For discrete action: softmax $\pi_\theta(s, a) \propto e^{\phi(s,a)^\top \theta}$
        - For continuous action: Gaussian $a \sim \mathcal{N}(\mu(s), \sigma^2)$

Direct Policy Learning

Advantages:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Disadvantages:

- Typically converge to a local rather than global optimum
- Evaluating a policy is typically inefficient and high variance

Objective Function

- Goal: given policy $\pi_\theta(s, a)$ with parameters θ, find best θ
- But how do we measure the quality of a policy π_θ?
- In episodic environments we can use the start value

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

- In continuing environments we can use the average value

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

- Or the average reward per time-step

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \mathcal{R}_s^a$$

- where $d^{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ

Objective Function

- Goal: given policy $\pi_\theta(s, a)$ with parameters θ, find best θ
- But how do we measure the quality of a policy π_θ?
- In episodic environments we can use the start value

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

- In continuing environments we can use the average value

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

- Or the average reward per time-step

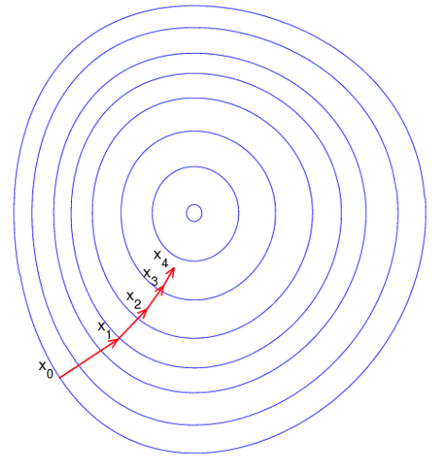
$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \mathcal{R}_s^a$$

- where $d^{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ

The challenge is that the distribution can only be estimated when the agent itself samples data.

Policy Gradient

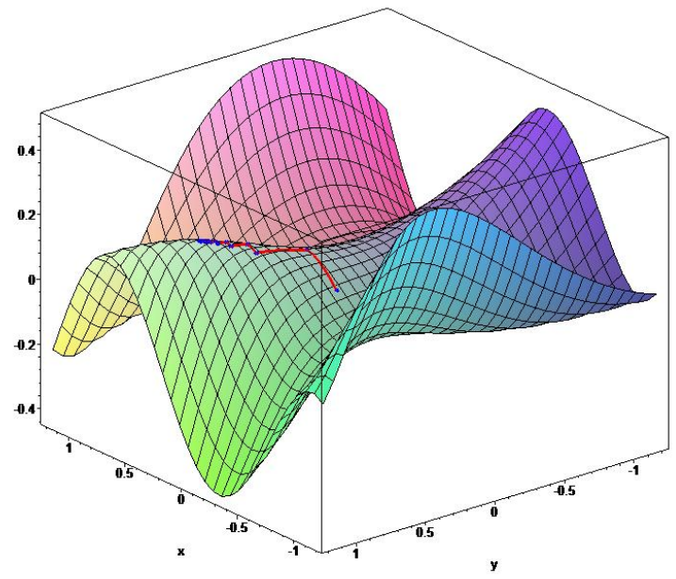
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a *local* maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ



$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

- Where $\nabla_\theta J(\theta)$ is the policy gradient

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$



- and α is a step-size parameter

Policy Gradient Theorem

Policy Gradient Methods for Reinforcement Learning with Function Approximation

Richard S. Sutton, David McAllester, Satinder Singh, Yishay Mansour
AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932

Theorem

For any differentiable policy $\pi_\theta(s, a)$,
for any of the policy objective functions $J = J_1, J_{avR}$, *or* $\frac{1}{1-\gamma} J_{avV}$,
the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(s, a) \; Q^{\pi_\theta}(s, a) \right]$$

Policy Gradient Theorem

Policy Gradient Methods for Reinforcement Learning with Function Approximation

Richard S. Sutton, David McAllester, Satinder Singh, Yishay Mansour
AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932

Theorem

For any differentiable policy $\pi_\theta(s, a)$,
for any of the policy objective functions $J = J_1, J_{avR}$, *or* $\frac{1}{1-\gamma} J_{avV}$,
the policy gradient is

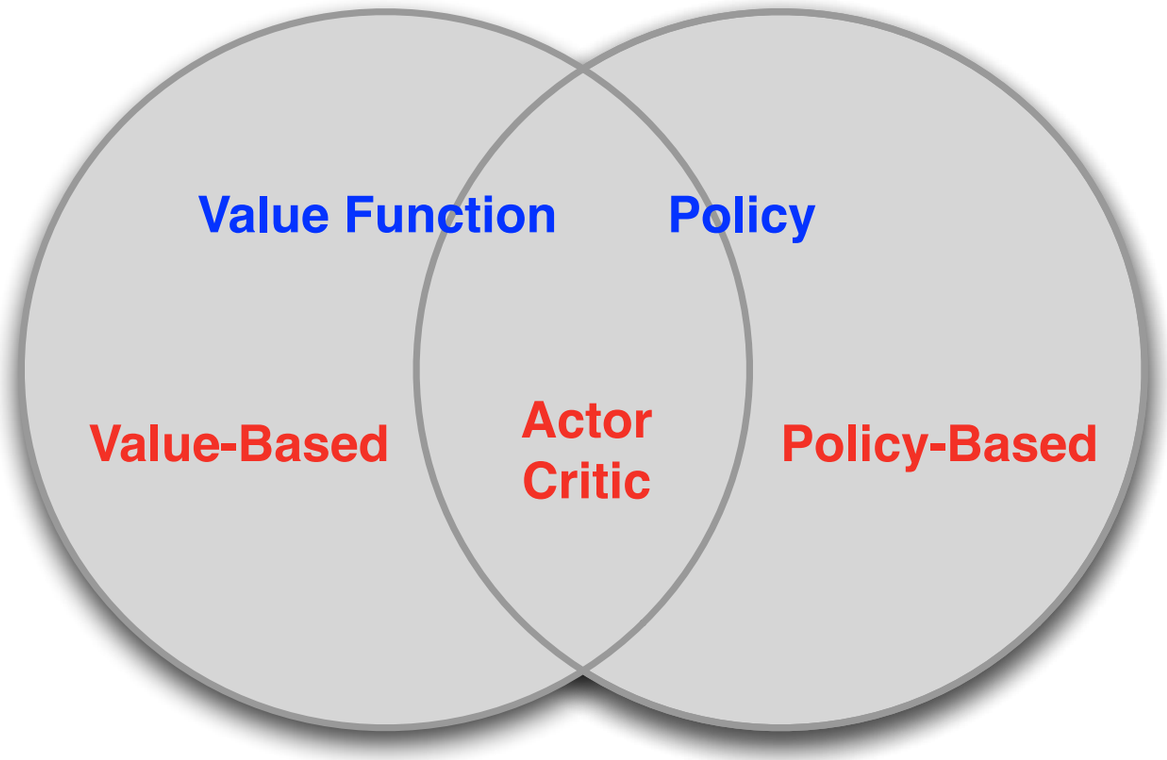
$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(s, a) \, Q^{\pi_\theta}(s, a) \right]$$

Need to estimate value function

Monte-Carlo or temporal difference

Actor-Critic

- Value Based
  - Learnt Value Function
  - Implicit policy (e.g. ϵ-greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy



Machine Learning:VI

- Reinforcement learning
  - Review of Basics
  - Function approximation
  - Policy gradient & actor-critic methods
- Deep reinforcement learning
- Integrating learning and planning
- Take-home messages

Practical Issues for RL

- Reward design: an art
- State feature design: also an art
- The environment is too complicated to model.

- Can we apply deep learning to RL?
- Use deep network to represent value function / policy / model
- Optimise value function / policy /model end-to-end
- Using stochastic gradient descent

Deep Q-Network

- Represent value function by deep Q-network with weights w

$$Q(s, a, w) \approx Q^{\pi}(s, a)$$

- Define objective function by mean-squared error in Q-values

$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

- Leading to the following Q-learning gradient

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

- Optimise objective end-to-end by SGD, using $\frac{\partial L(w)}{\partial w}$

Recall function approximation in P. 47.

Stability Issues for Deep RL

Naive Q-learning oscillates or diverges with neural nets

1. Data is sequential
   - Successive samples are correlated, non-iid
2. Policy changes rapidly with slight changes to Q-values
   - Policy may oscillate
   - Distribution of data can swing from one extreme to another
3. Scale of rewards and Q-values is unknown
   - Naive Q-learning gradients can be large unstable when backpropagated

The bias and instability issue we have discussed.
More severe for NNs.

DQN Techs

DQN provides a stable solution to deep value-based RL

1. Use experience replay
   - Break correlations in data, bring us back to iid setting
   - Learn from all past policies
2. Freeze target Q-network
   - Avoid oscillations
   - Break correlations between Q-network and target
3. Clip rewards or normalize network adaptively to sensible range
   - Robust gradients

Experience Replay

To remove correlations, build data-set from agent's own experience

- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Optimise MSE between Q-network and Q-learning targets, e.g.

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right)^2 \right]$$

off-policy!

Fixed Network

To avoid oscillations, fix parameters used in Q-learning target

- Compute Q-learning targets w.r.t. old, fixed parameters w^-

$$r + \gamma \max_{a'} Q(s', a', w^-)$$

- Optimise MSE between Q-network and Q-learning targets

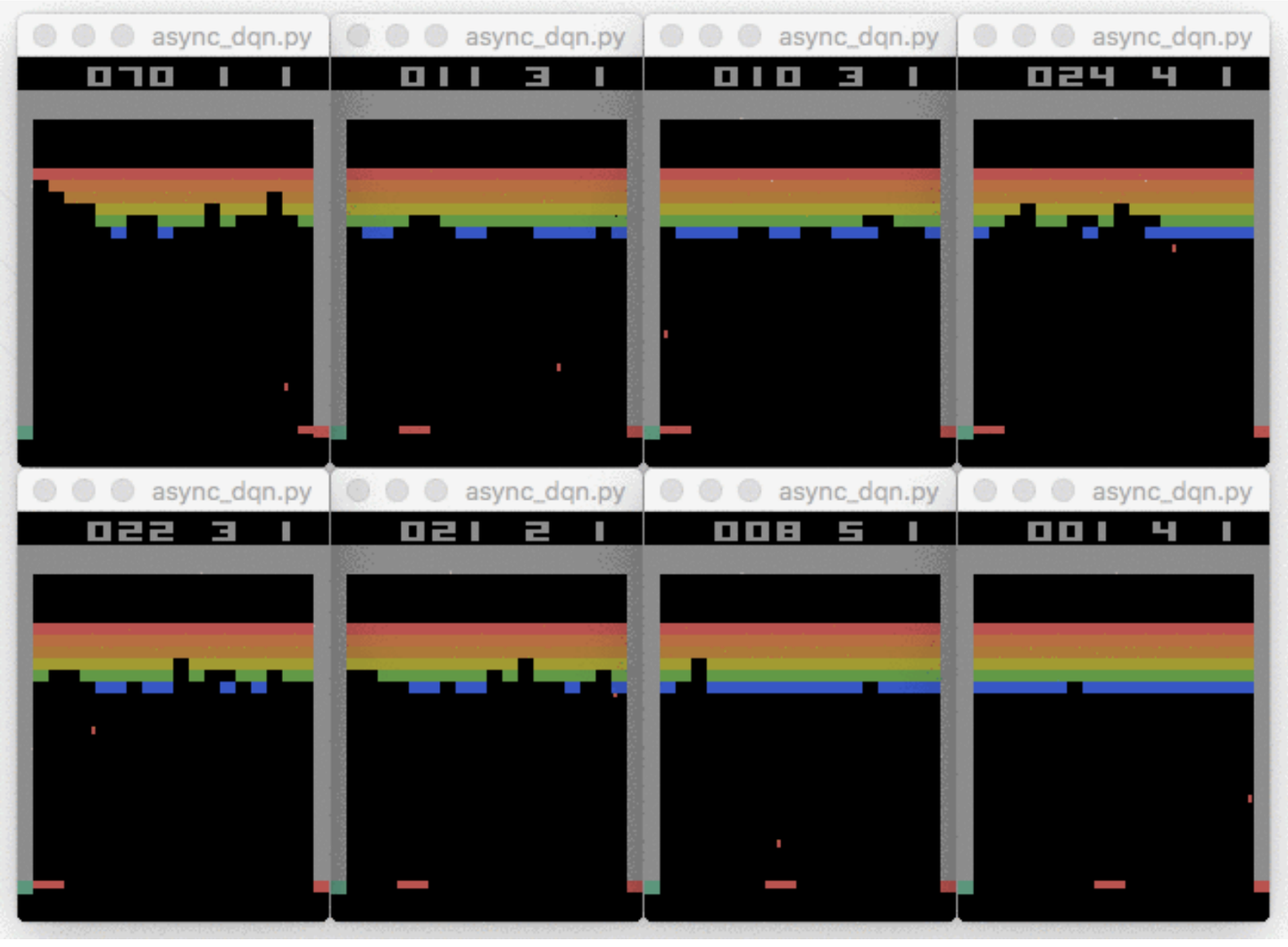
$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w) \right)^2 \right]$$

- Periodically update fixed parameters $w^- \leftarrow w$

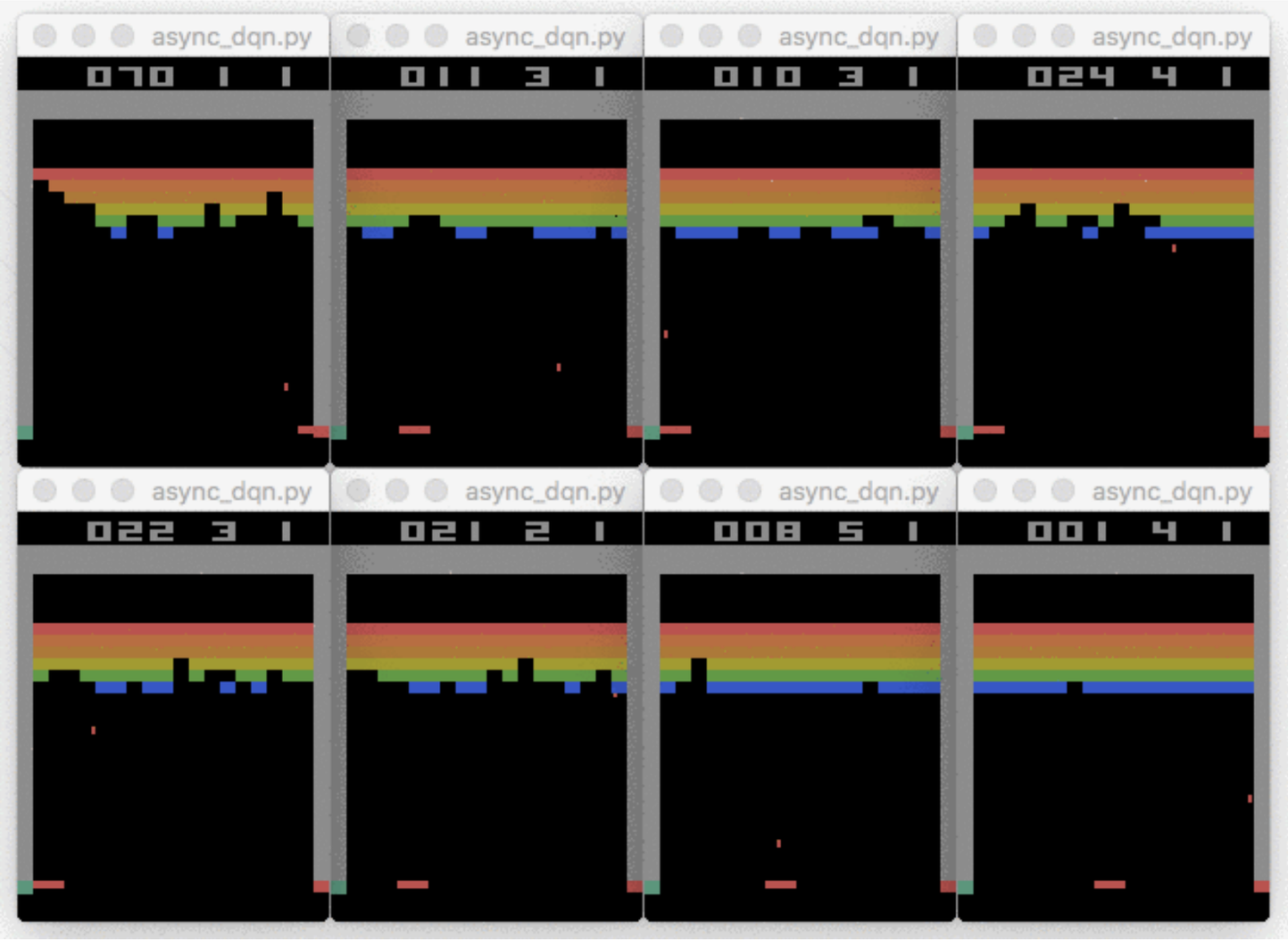
Clipped Rewards

- DQN clips the rewards to $[-1, +1]$
- This prevents Q-values from becoming too large
- Ensures gradients are well-conditioned
- Can't tell difference between small and large rewards

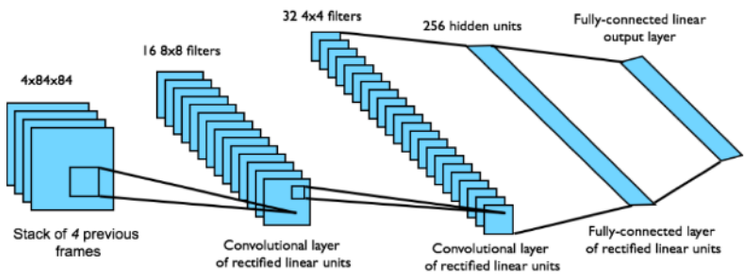
DQN for Atari Games



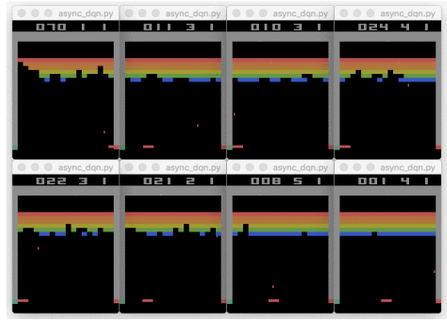
DQN for Atari Games



DQN for Atari Games



control



sampling
optimize

self-generated data

Replay Buffer

Published: 25 February 2015

Human-level control through deep reinforcement learning

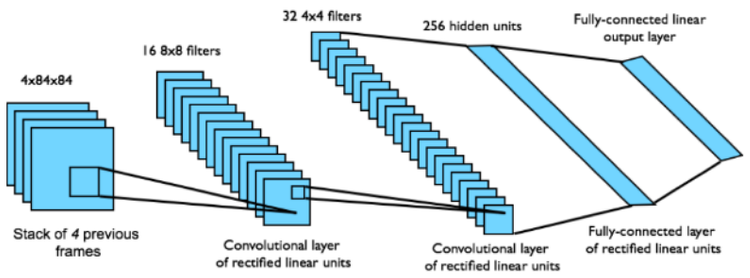
Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis

Nature **518**, 529–533 (2015) | Cite this article

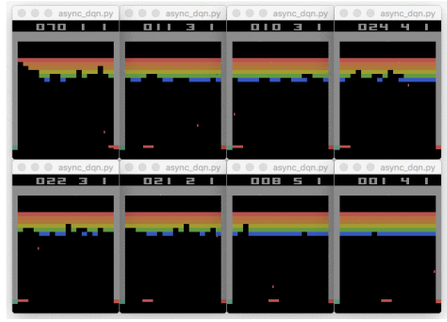
438k Accesses | **10525** Citations | **1546** Altmetric | Metrics

First break through of deep RL.

DQN for Atari Games



control



sampling
optimize

self-generated data

Replay Buffer

Published: 25 February 2015

Human-level control through deep reinforcement learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis

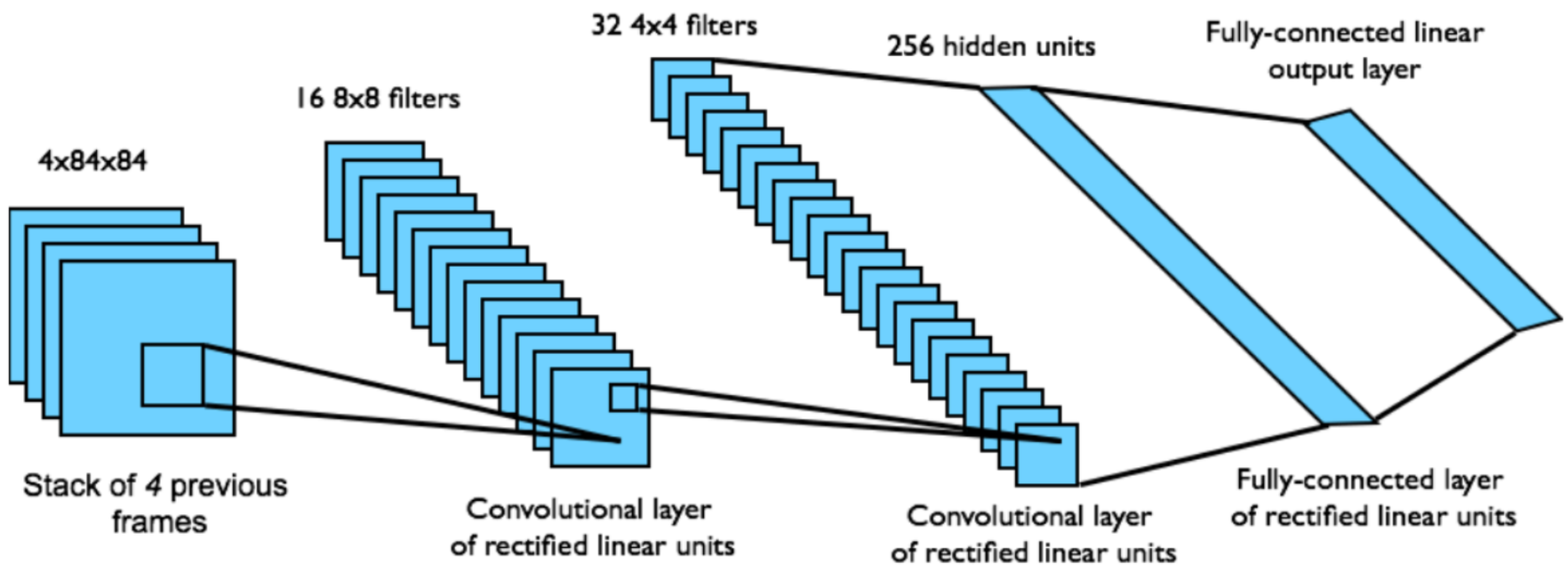
Nature **518**, 529–533 (2015) | Cite this article

438k Accesses | **10525** Citations | **1546** Altmetric | Metrics

First break through of deep RL.

DQN for Atari Games

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

[Mnih et al.]

Machine Learning:VI

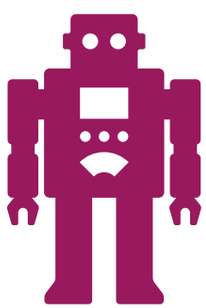
- Reinforcement learning
  - Review of Basics
  - Function approximation
  - Policy gradient & actor-critic methods
  - Deep reinforcement learning
- Integrating learning and planning
- Take-home messages

Internal and External Model

Models are crucial in decision making problems.
Whenever we have the external model or can obtain the internal model, we can combine the power of learning and planning (e.g. search, dynamic programming).

internal model

agent



partial state

utility

action

environment (external model)

Integrating Learning and Planning

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from real experience
- Model-Based RL (using Sample-Based Planning)
  - Learn a model from real experience
  - Plan value function (and/or policy) from simulated experience
- Dyna
  - Learn a model from real experience
  - Learn and plan value function (and/or policy) from real and simulated experience

Machine Learning Proceedings 1990

Proceedings of the Seventh International Conference, Austin, Texas, June 21–23, 1990

1990, Pages 216-224

Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming

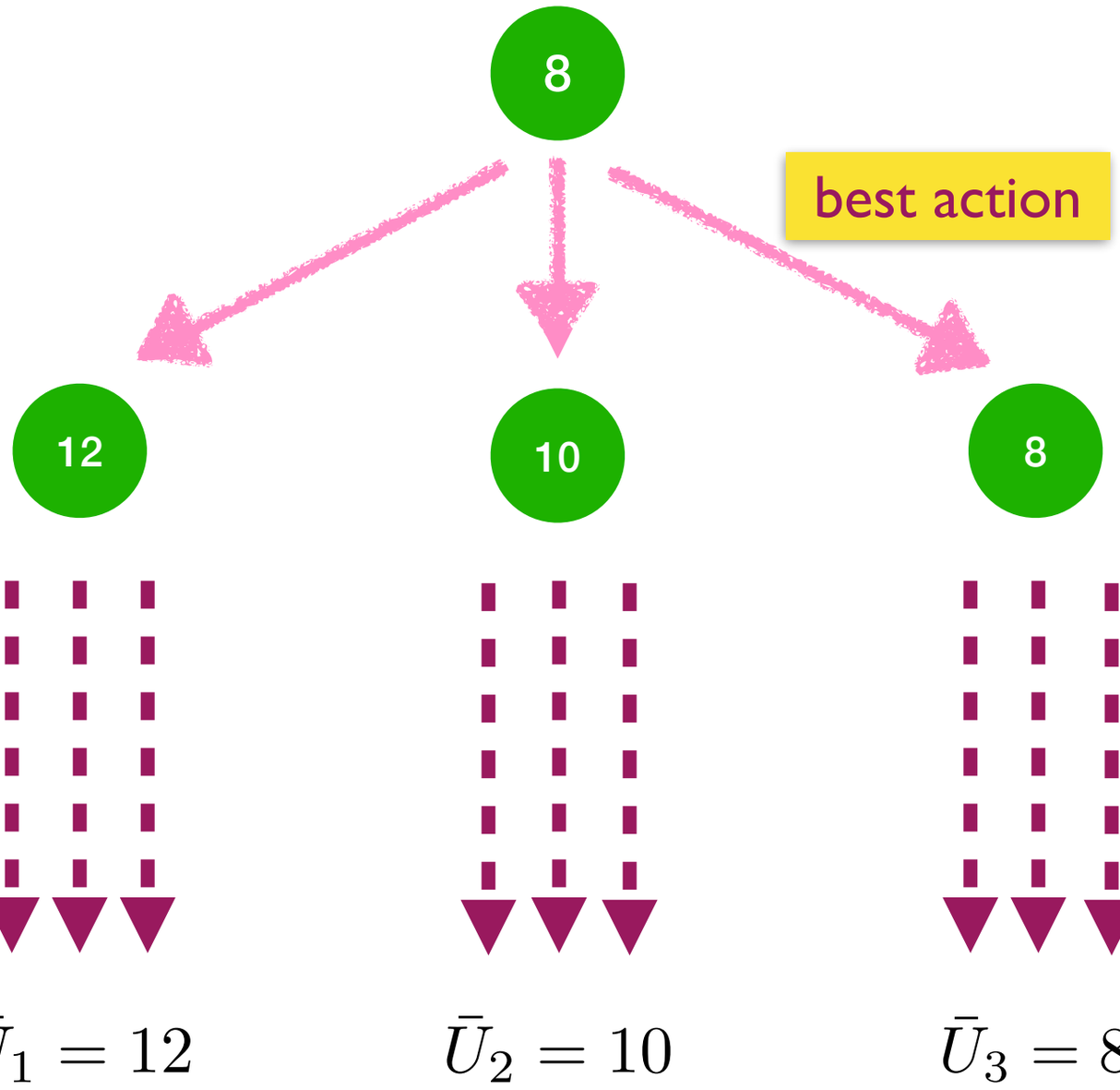
Richard S. Sutton

Planning by DP

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $q_\pi(s, a)$ or $q_*(s, a)$
- Complexity $O(m^2n^2)$ per iteration

Monte-Carlo Simulation

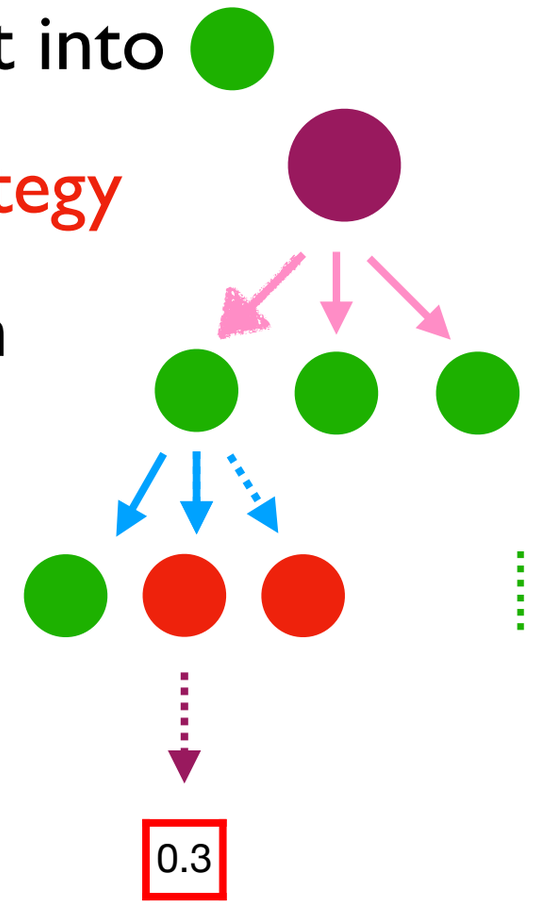


repeated random run

Converge to true utility when the #run is sufficient!
But in real game playing, the time and space for simulation is limited.
We need a smart strategy to decide the order of simulation.

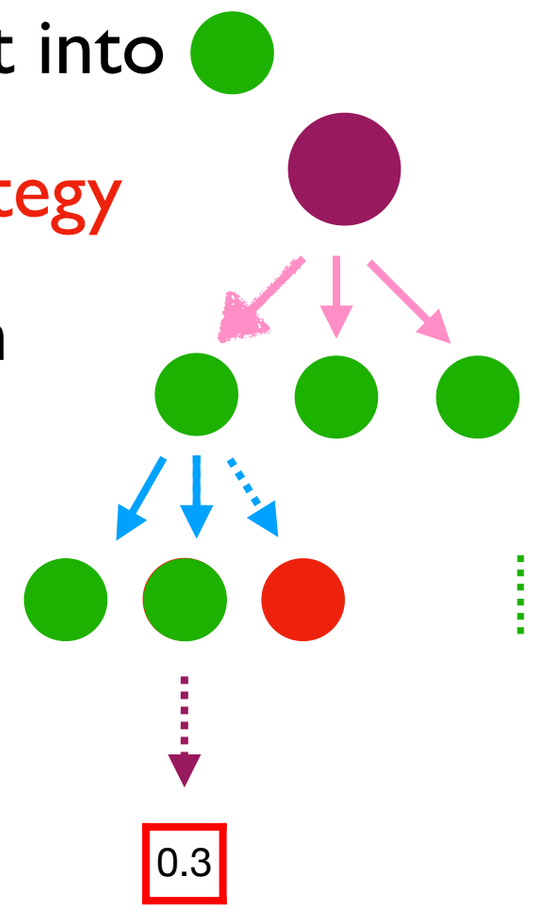
Monte-Carlo Tree Search (MCTS)

- Nodes of two kinds: ● visited node with unexpanded child, and ● unvisited node.
- During MCTS, we use two strategies: tree and default
- Algorithm: repeat until time or space limit:
  - Selection: choose one node among ● using tree strategy
  - Expansion: expand an unvisited child and put into ●
  - Simulation: simulate down using default strategy
  - Update: update MC estimation through path
- Output the best action to play



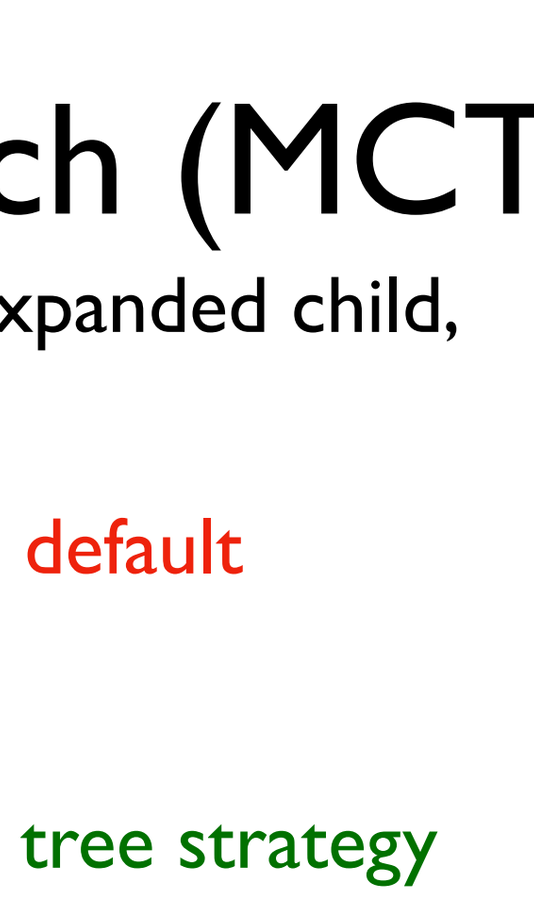
Monte-Carlo Tree Search (MCTS)

- Nodes of two kinds: ● visited node with unexpanded child, and ● unvisited node.
- During MCTS, we use two strategies: tree and default
- Algorithm: repeat until time or space limit:
  - Selection: choose one node among ● using tree strategy
  - Expansion: expand an unvisited child and put into ●
  - Simulation: simulate down using default strategy
  - Update: update MC estimation through path
- Output the best action to play



Monte-Carlo Tree Search (MCTS)

- Nodes of two kinds: ● visited node with unexpanded child, and ● unvisited node.
- During MCTS, we use two strategies: tree and default
- Algorithm: repeat until time or space limit:
  - Selection: choose one node among ● using tree strategy
  - Expansion: expand an unvisited child and put into ●
  - Simulation: simulate down using default strategy
  - Update: update MC estimation through path
- Output the best action to play



The default strategy is usually random play
The tree strategy is essential:
Deciding the order of search

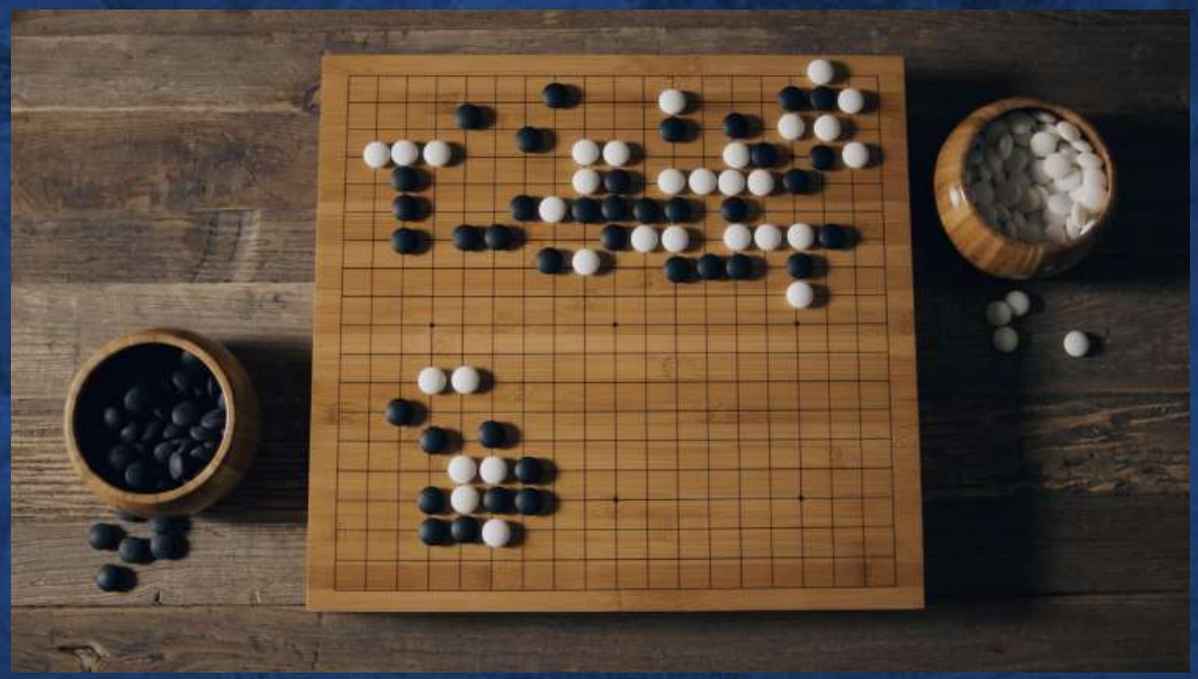
AlphaGo:
Integration of Learning & Planning

Why is Go hard for computers to play?

Game tree complexity = b^d

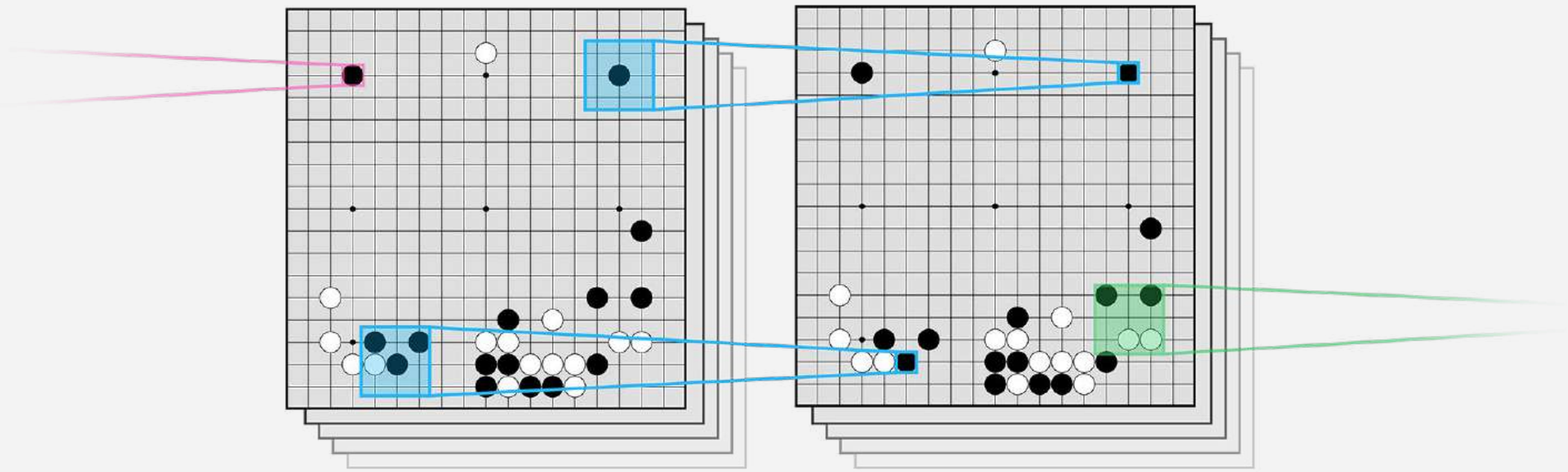
Brute force search intractable:

1. Search space is huge
2. “Impossible” for computers to evaluate who is winning



Google DeepMind

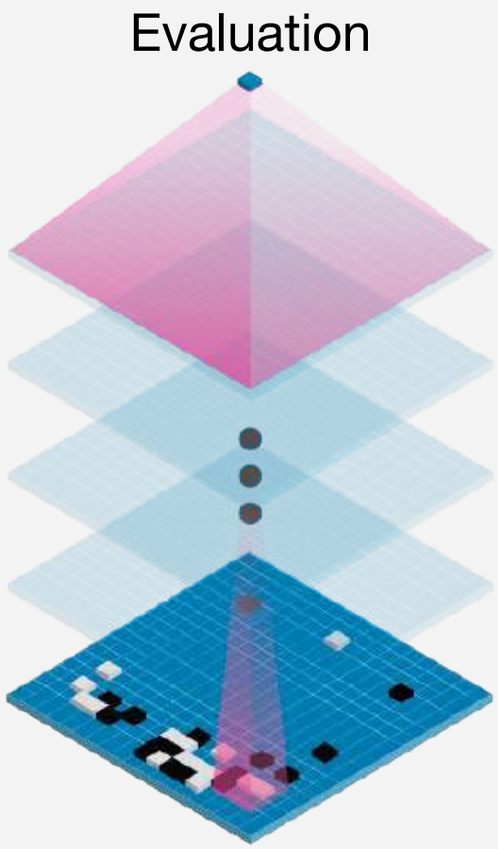
Convolutional neural network

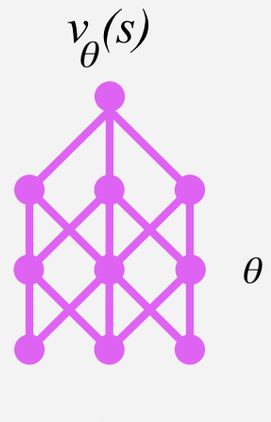


Google DeepMind

AlphaGo introduces three conv. nets for learning.
Use images of board as state input.

Value network



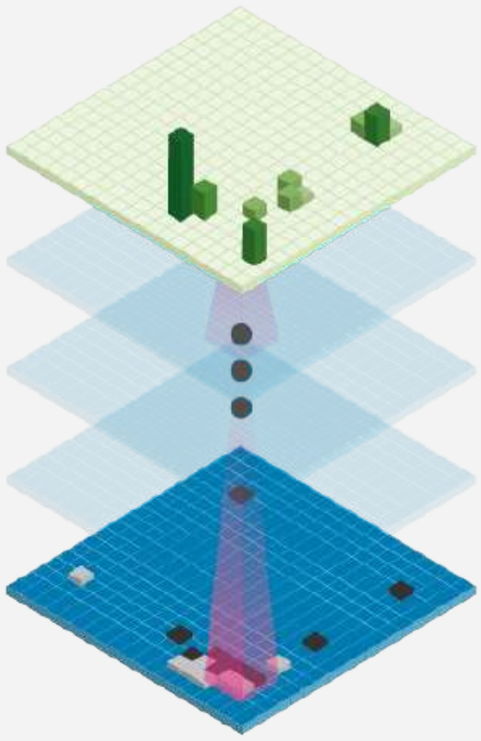


Google DeepMind

The value function of RL.

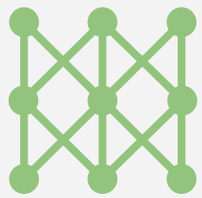
Policy network

Move probabilities



Position

$p_\sigma(a|s)$



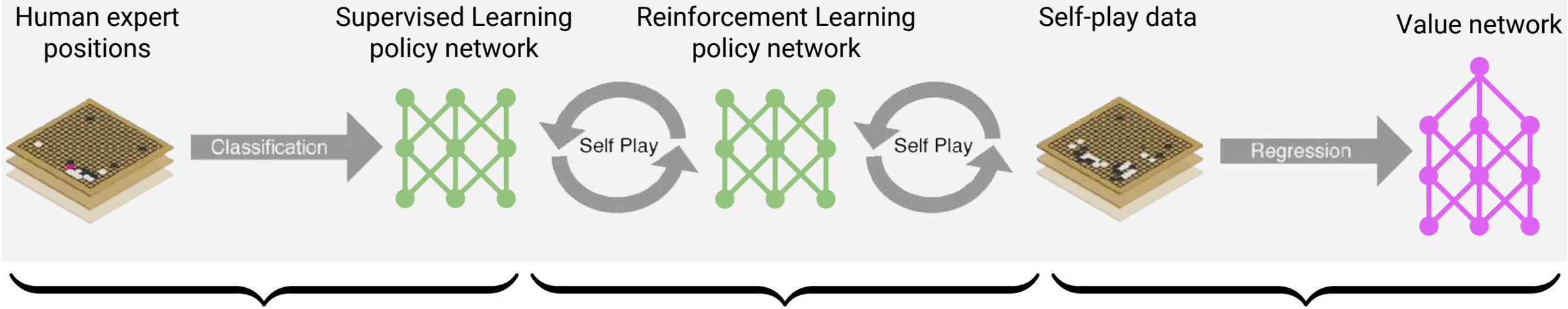
σ

s

Google DeepMind

Policy network of RL.

Neural network training pipeline



First stage:
Supervised classification

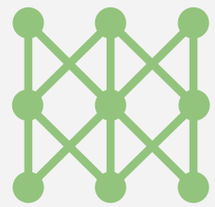
Second stage:
Policy gradient RL

Second stage:
Supervised regression

Learning: First Stage

Supervised learning of policy networks

Policy network: 12 layer convolutional neural network



Training data: 30M positions from human expert games (KGS 5+ dan)

Training algorithm: maximise likelihood by stochastic gradient descent

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma}$$

Training time: 4 weeks on 50 GPUs using Google Cloud

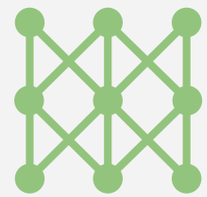
Results: 57% accuracy on held out test data (state-of-the art was 44%)

Google DeepMind

Learning: Second Stage

Reinforcement learning of policy networks

Policy network: 12 layer convolutional neural network



Training data: games of self-play between policy network

Training algorithm: maximise wins z by policy gradient reinforcement learning

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma} z$$

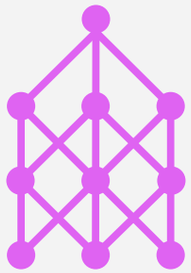
Training time: 1 week on 50 GPUs using Google Cloud

Results: 80% vs supervised learning. Raw network ~3 amateur dan.

Google DeepMind

Learning: Third Stage

Reinforcement learning of value networks



Value network: 12 layer convolutional neural network

Training data: 30 million games of self-play

Training algorithm: minimise MSE by stochastic gradient descent

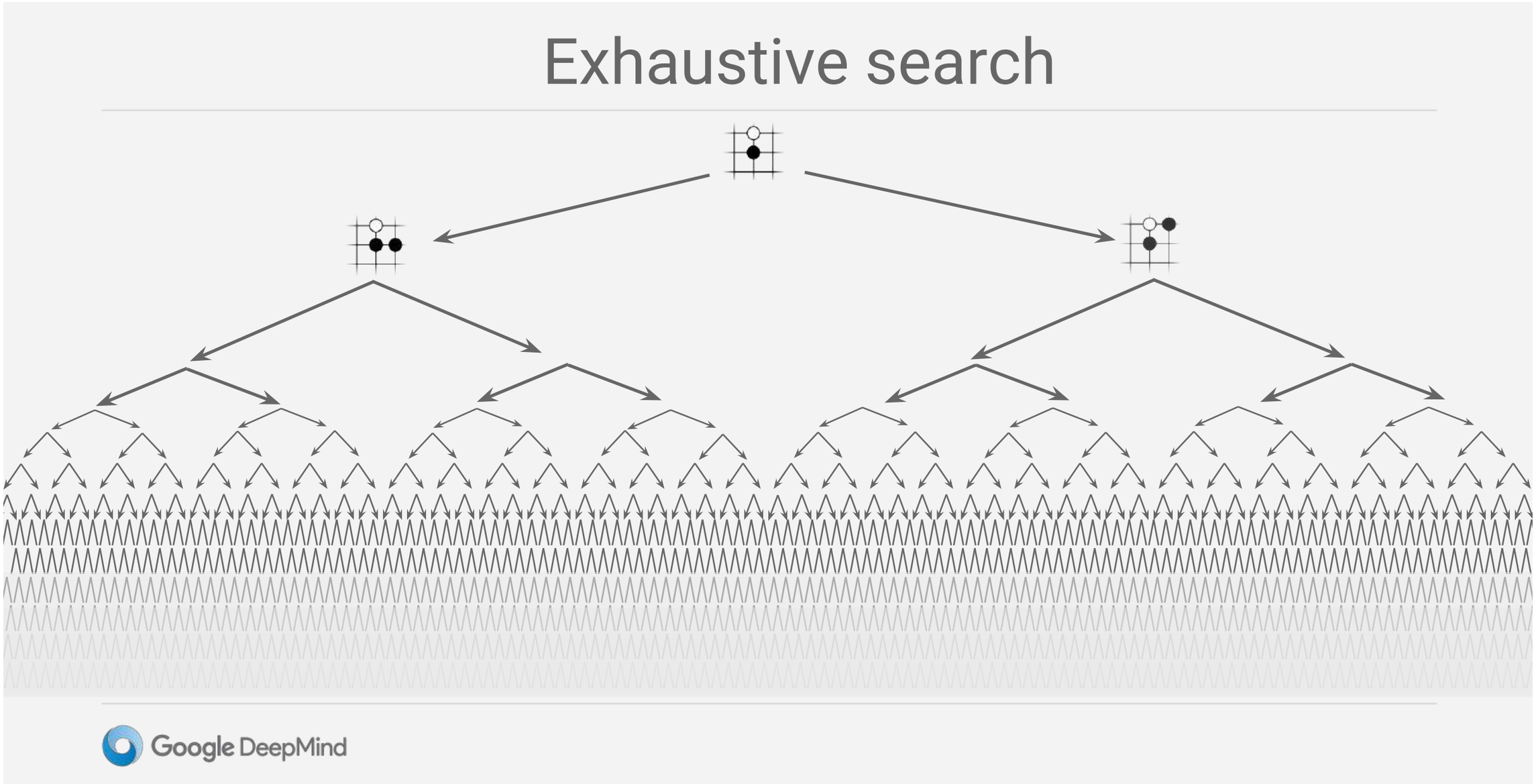
$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s))$$

Training time: 1 week on 50 GPUs using Google Cloud

Results: First strong position evaluation function - previously thought impossible

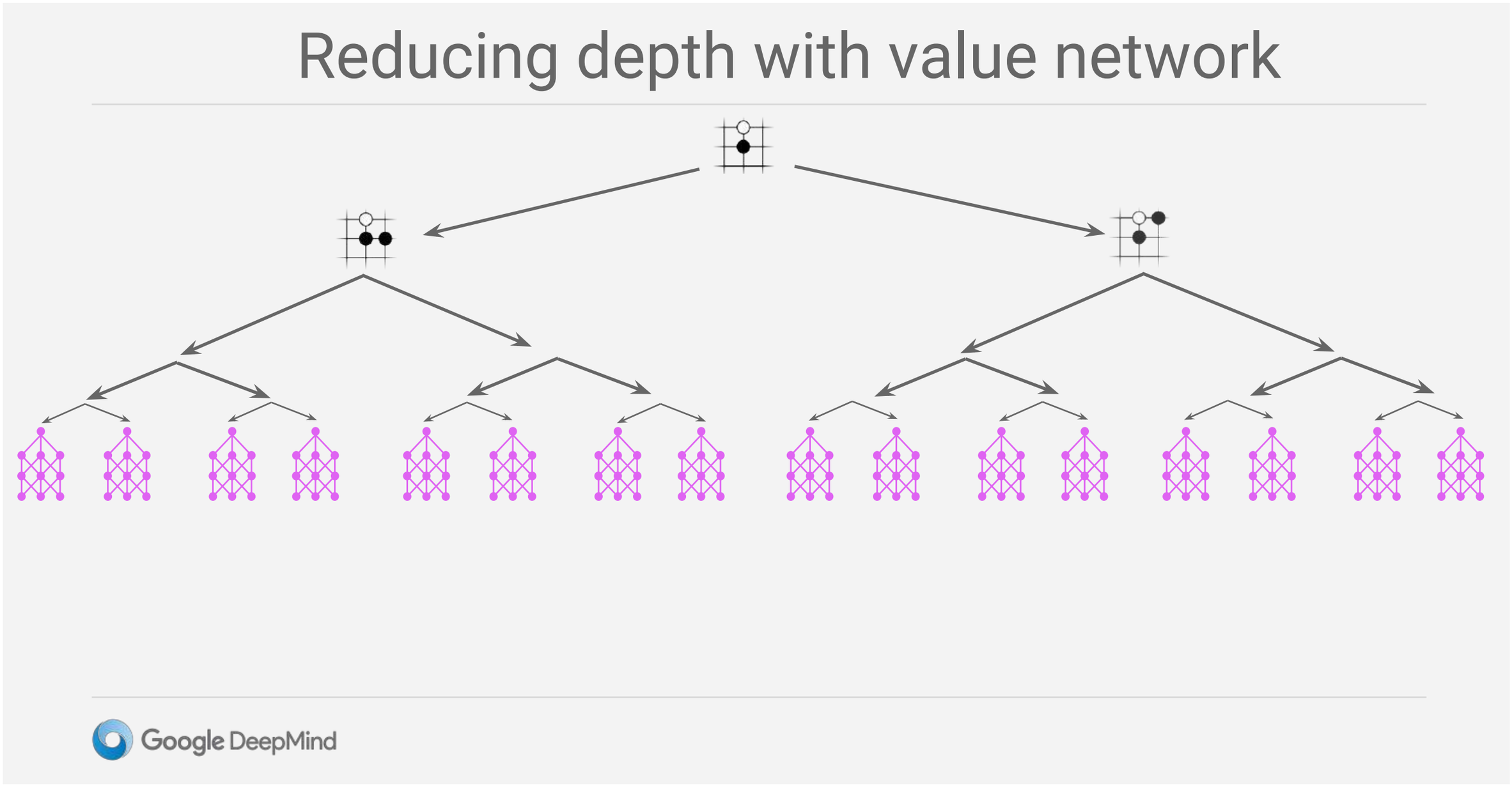
Google DeepMind

Real Play: MCTS



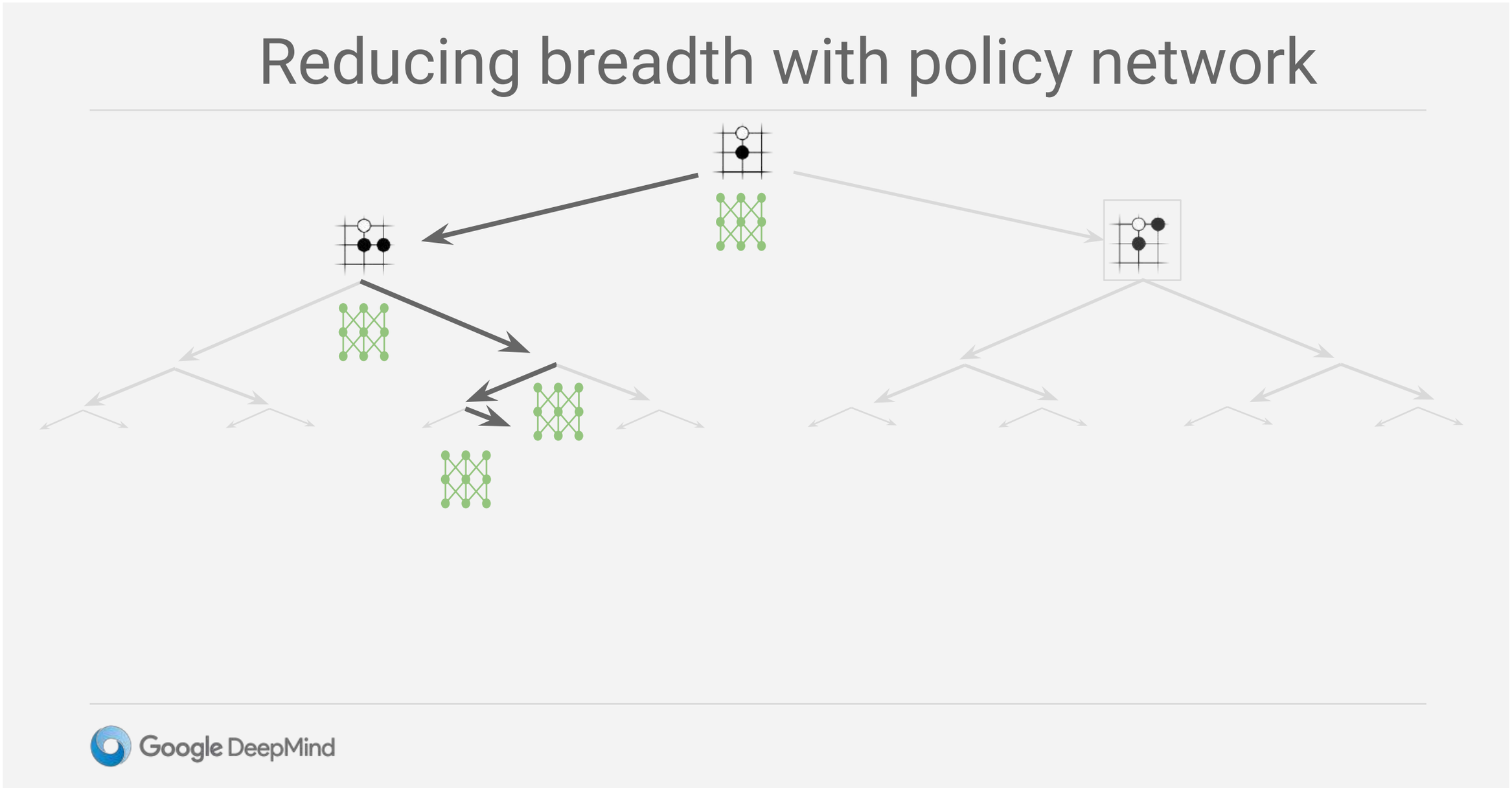
Two key steps:
node expansion and repeated random simulation

Real Play: MCTS



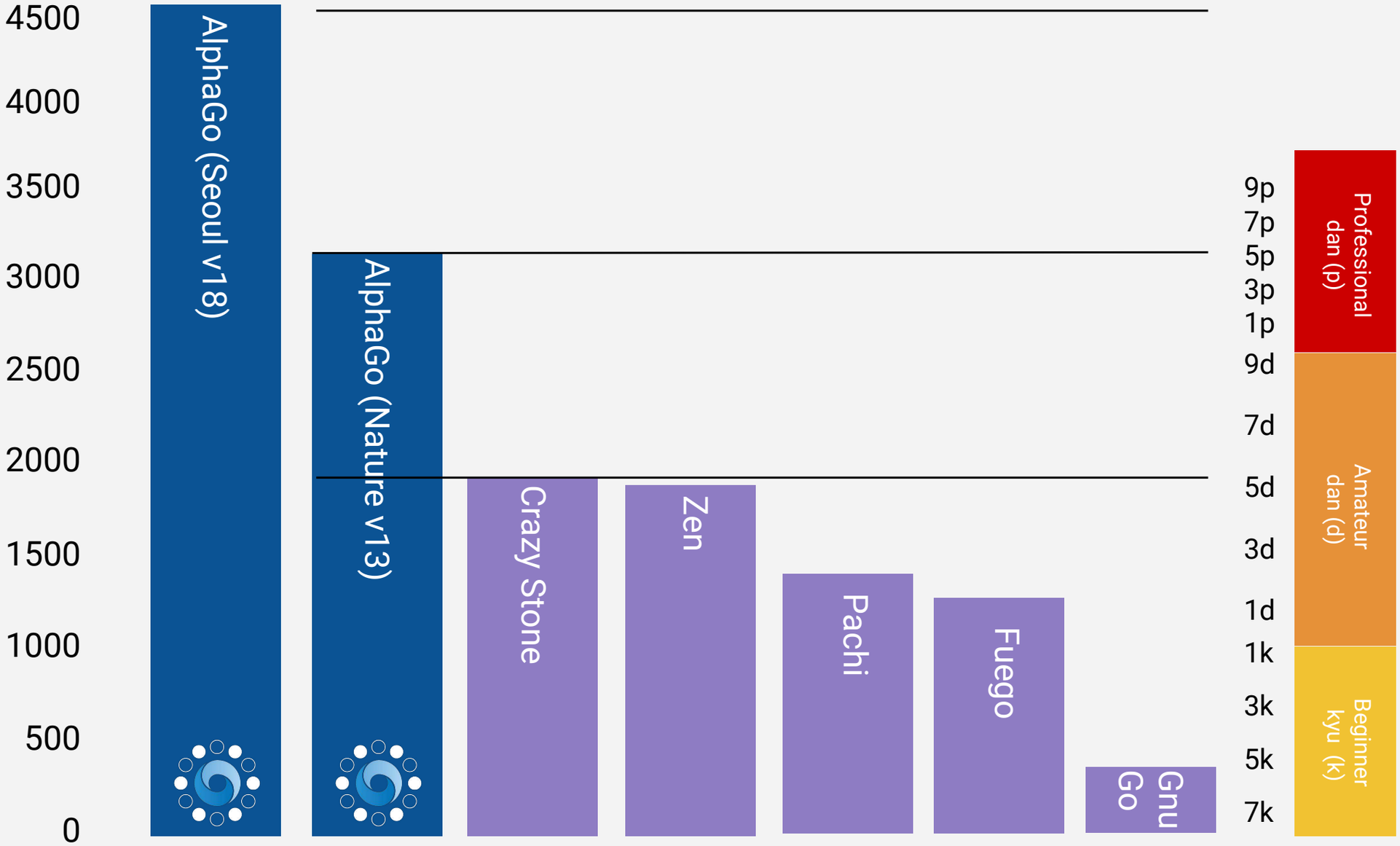
With value network, we can expand fewer depth
since the value of nodes can also be obtained from the value network.

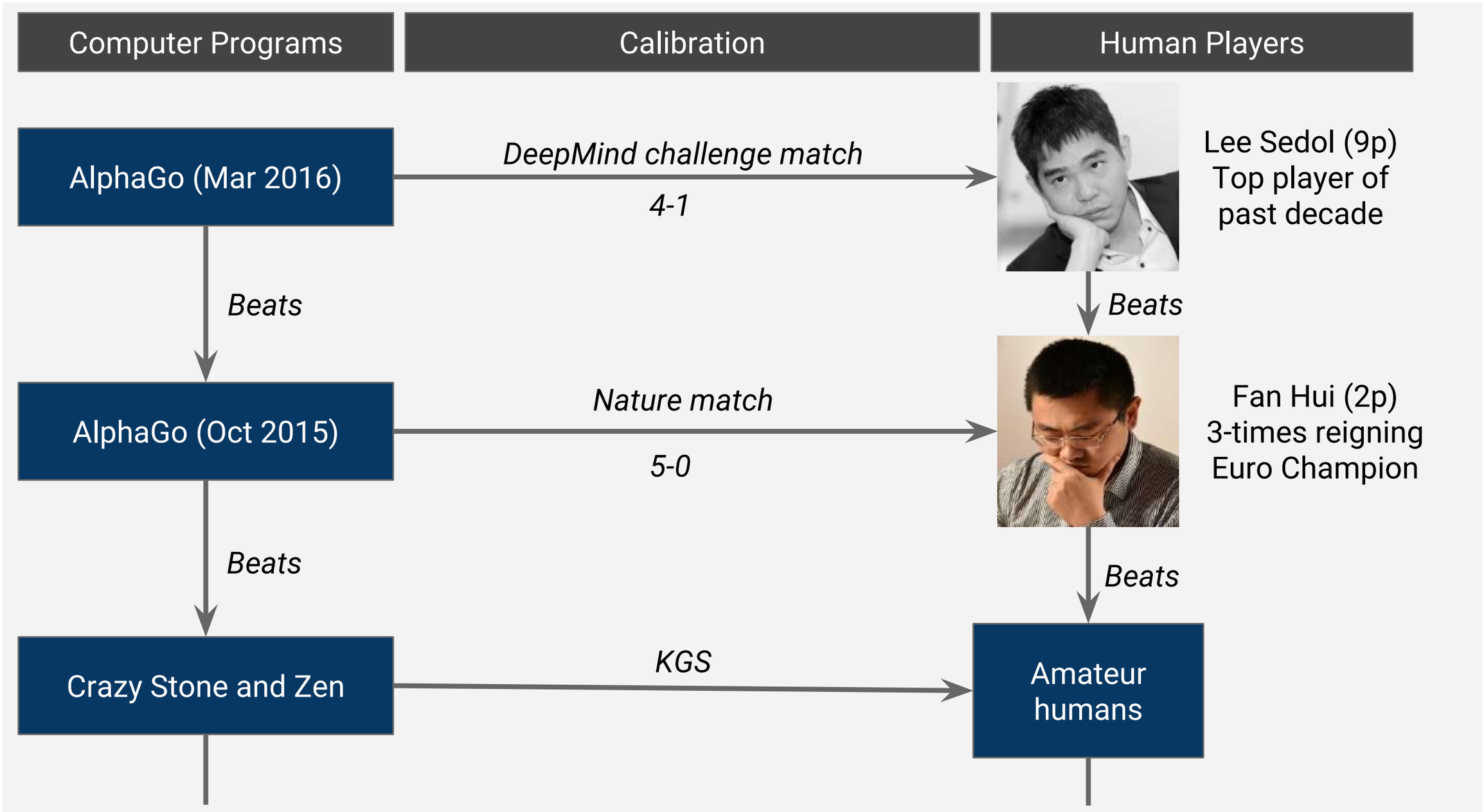
Real Play: MCTS



With policy network, we can simulate with fewer times
since the simulation can be guided by policy network over random play.

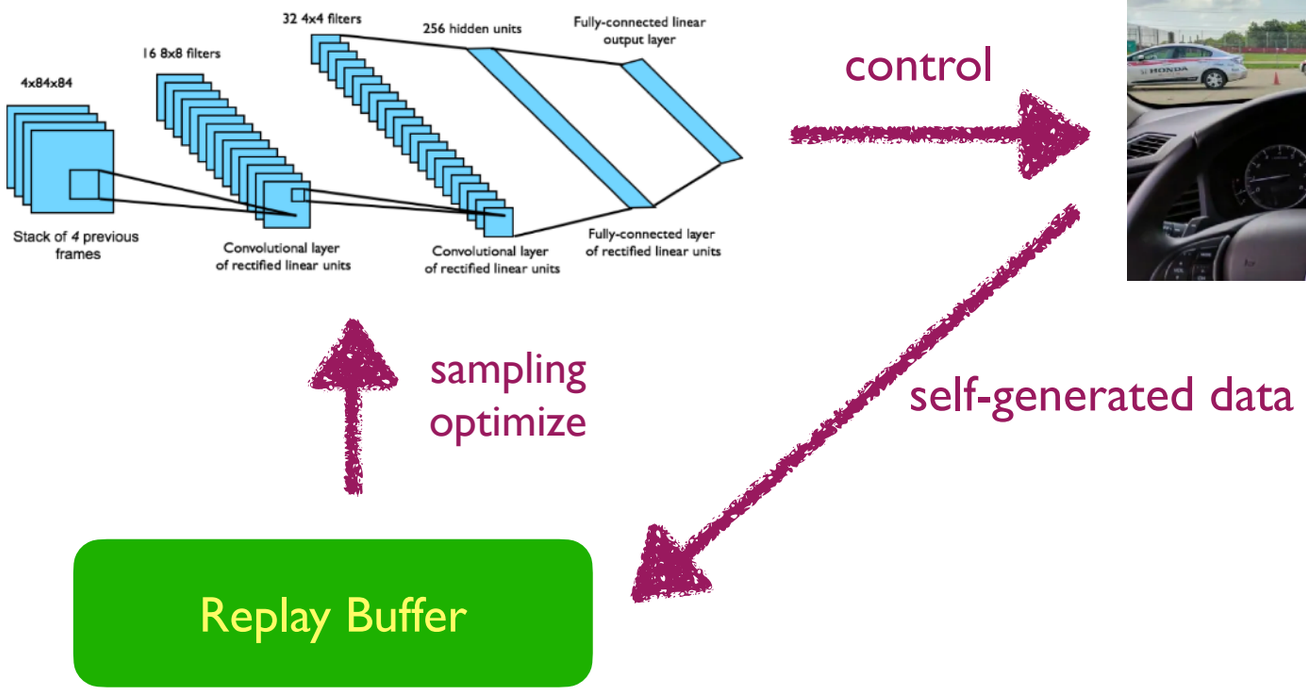
Evaluating AlphaGo against computers




Computer Programs
Calibration
Human Players
AlphaGo (Mar 2016)
DeepMind challenge match
4-1
Lee Sedol (9p)
Top player of
past decade
Beats
Beats
AlphaGo (Oct 2015)
Nature match
5-0
Fan Hui (2p)
3-times reigning
Euro Champion
Beats
Beats
Crazy Stone and Zen
KGS
Amateur
humans

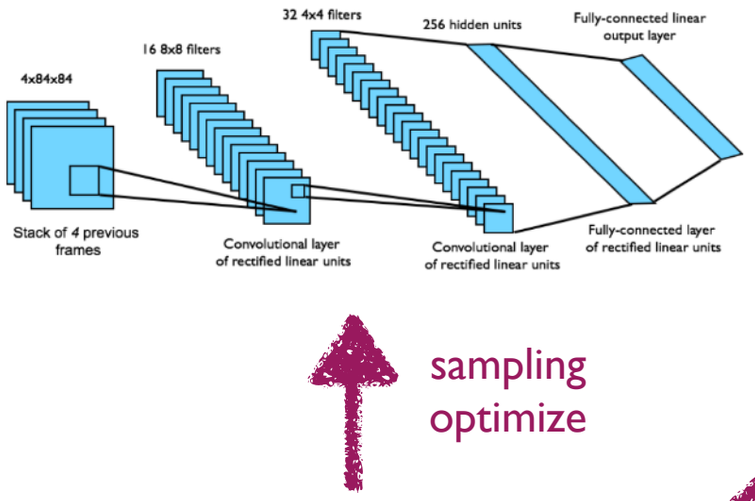
Why hasn't DRL solved many real-world problems?

- Deep RL usually does not use (learn) a world model.



Why hasn't DRL solved many real-world problems?

- Deep RL usually does not use (learn) a world model.



control

sampling
optimize

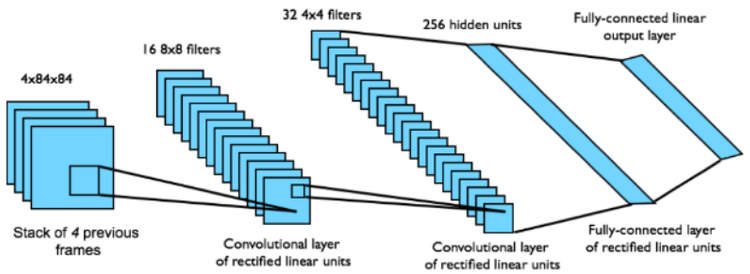
self-generated data

?

Replay Buffer

Why hasn't DRL solved many real-world problems?

- Deep RL usually does not use (learn) a world model.



control

sampling
optimize

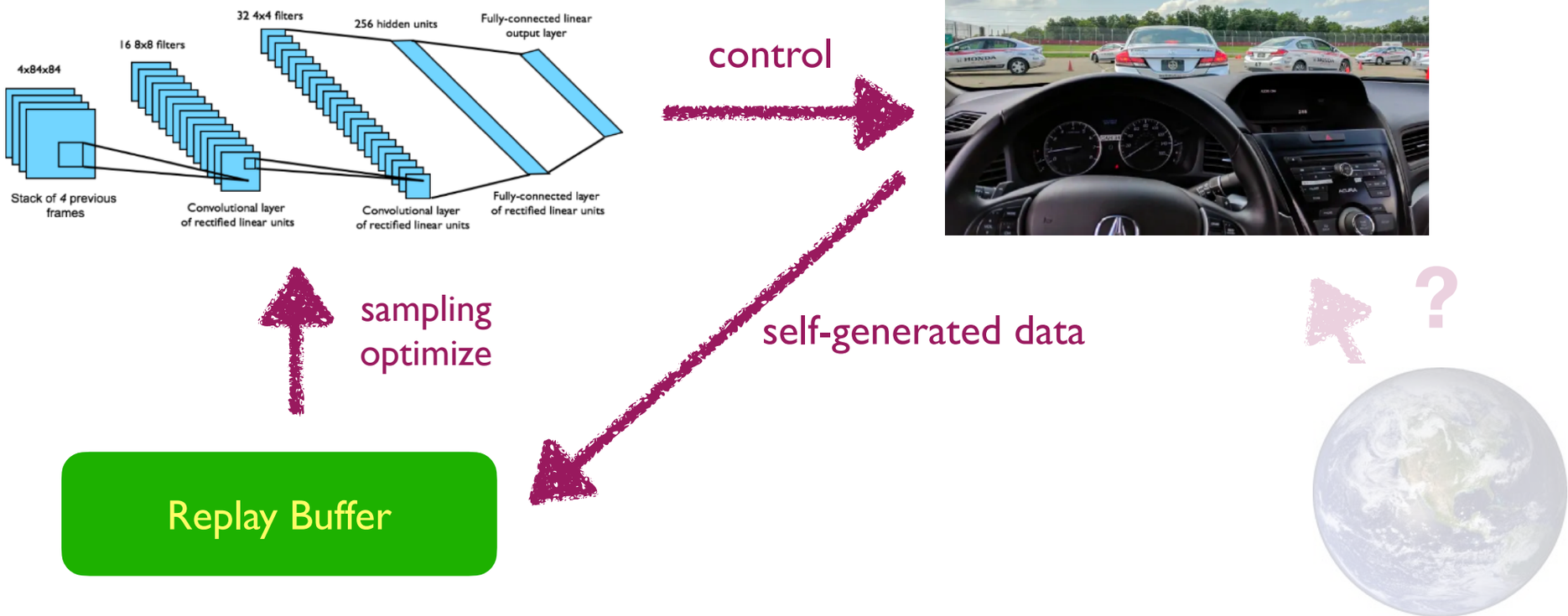
self-generated data

Replay Buffer

?

Why hasn't DRL solved many real-world problems?

- Deep RL usually does not use (learn) a world model.



Without a world model, learning from self-generated data requires many trial-and-errors in the real world.
Large cost. Bad generalization to new task.

Model-Based RL

Scenarios of decision making:

Model-Based RL

Scenarios of decision making:

- Planning: directly solve the optimal policy when the world model is known (dynamic programming, Monte-Carlo tree search…)

Model-Based RL

Scenarios of decision making:

- Planning: directly solve the optimal policy when the world model is known (dynamic programming, Monte-Carlo tree search…)

- Model-free RL: learn by trial-and-error (Q-learning, policy gradient, actor-critic…)

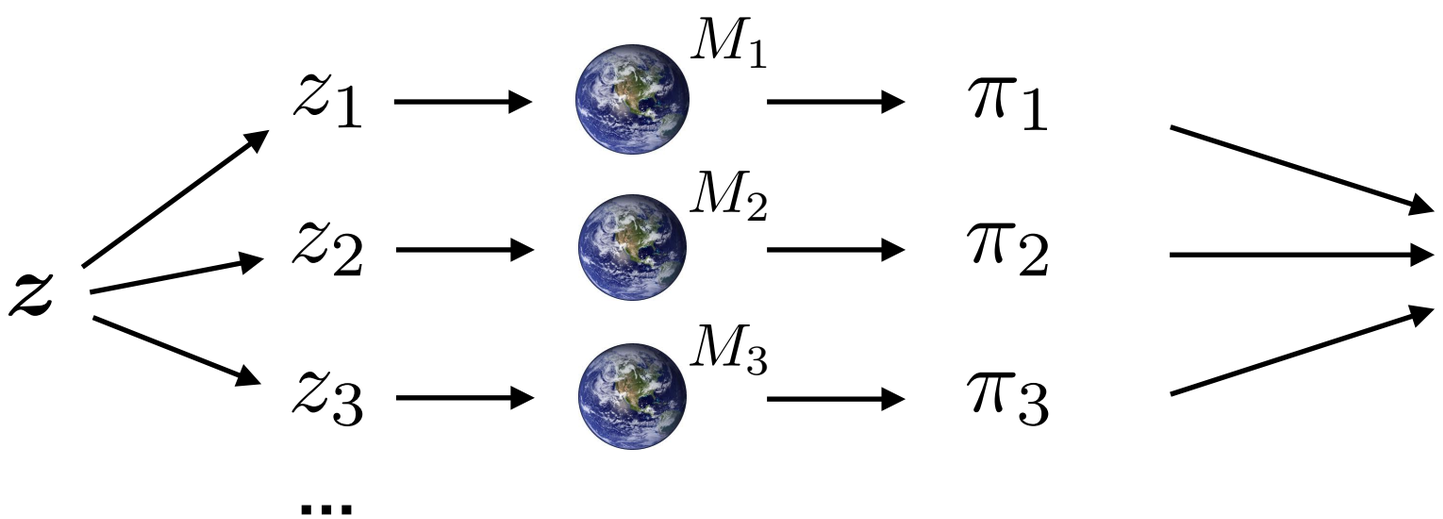
Model-Based RL

Scenarios of decision making:

- Planning: directly solve the optimal policy when the world model is known (dynamic programming, Monte-Carlo tree search…)

- Model-free RL: learn by trial-and-error (Q-learning, policy gradient, actor-critic…)

- Model-based RL: learn the world model during learning, do RL or planning using the model.



policy learned with low cost & robust to world change

Model-Based RL

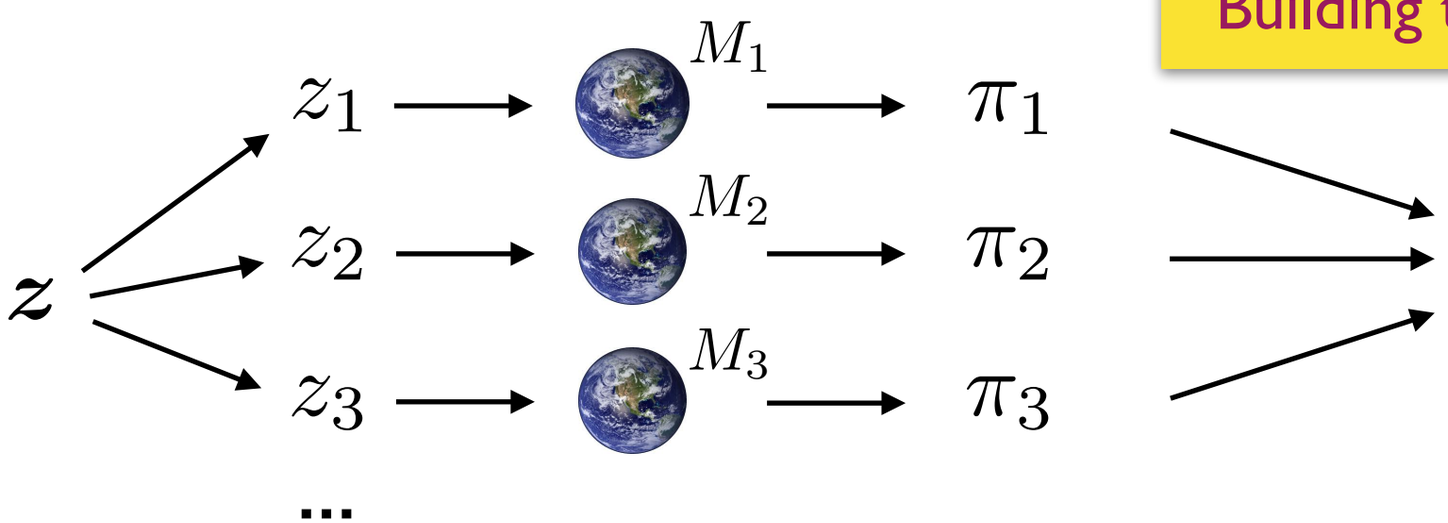
Scenarios of decision making:

- Planning: directly solve the optimal policy when the world model is known (dynamic programming, Monte-Carlo tree search…)

- Model-free RL: learn by trial-and-error (Q-learning, policy gradient, actor-critic…)

- Model-based RL: learn the world model during learning, do RL or planning using the model.

Building the internal model is essential.



policy learned with low cost & robust to world change

Machine Learning:VI

- Reinforcement learning
  - Review of Basics
  - Function approximation
  - Policy gradient & actor-critic methods
  - Deep reinforcement learning
  - Integrating learning and planning
- Take-home messages

Take-Home Messages

- To solve large-scale RL problems, functional approximation of value functions or policies is essential.

- Policy gradient & actor-critic: direct learning of policies. Usually more efficient for deep RL.

- Integrating learning and planning is a powerful strategy for model-based decision-making.

- Next-step of RL: addressing the high-cost of interaction with environment, which is to building internal models.

Next week: summary and review.

Thanks for your attention!
Discussions?

Acknowledgement: Many materials in this lecture are taken from
https://www.davidsilver.uk/teaching/
https://www.davidsilver.uk/wp-content/uploads/2020/03/AlphaGo-tutorial-slides_compressed.pdf